Version of this document

Yag and LUA

General behavior

The structures
- Character sheet (cs)
- Pawn sheet (ps)
- Dice results (dr)
- dice roll origin (dro)

The yag object (LUA to Yag)
- yag.LuaDir
- yag.RollDice(f1, f2, ui, us)
- yag.ClearDice()
- yag.PublicMessage(s)
- yag.LocalMessage(s)
- yag.GetPawnInfo(id)
- yag.UpdatePawn(ids, ps)

The LUA functions called by Yag (Yag to LUA)
- Yag_CharacterSheetUpdate(sp, cs)
- Yag_RollDiceButton(sp, cs, id, f1, f2, dro)
- Yag_RollDiceCallback(sp, cs, id, dr1, dr2, ui, us)

Version of this document

If you found this document in your Yag installation, you may want to download the latest version at the following address: http://yagame.fr/lua-api/

Yag and LUA

One goal of Yag is to remains as general as possible about game rules.
This raises the need to customize it according to any specific game.
But Yag is coded in c++ and its guts are hidden from the player.
A common response to this problem is to embed in the host program (Yag) a scripting language such as LUA.

In order to exchange data, some of the internals of Yag must be exposed to LUA:
- Some Yag functions and variables must be accessed by the players through LUA
- Yag must be able to call some LUA functions

The present document describes the set of tools currently available to do so.

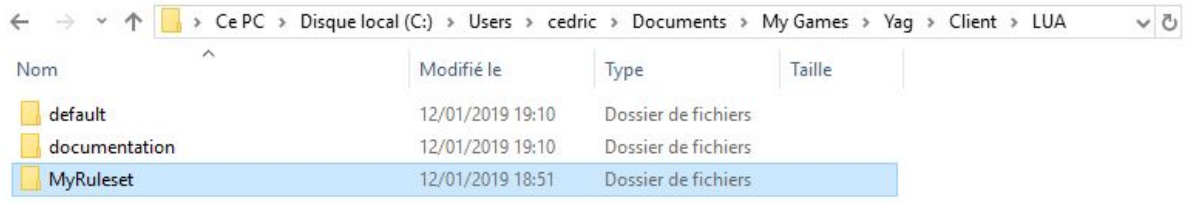
The embedded version of LUA is v5.3.

General behavior

To call some LUA code, Yag uses the content of a directory that must:

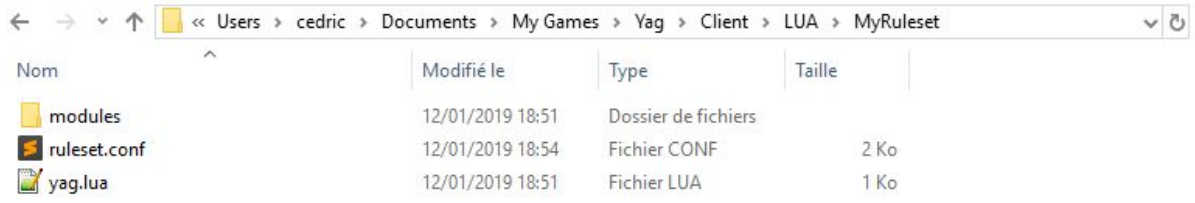
- Exist in <my documents>/My Games/Yag/Client/LUA
- Have a unique name specified in the LUA panel in the Yag UI.

For example, if the name of the directory is MyRuleSet, it must be created like so:
<my documents>/My Games/Yag/Client/LUA/MyRuleSet



And be set in Yag here:

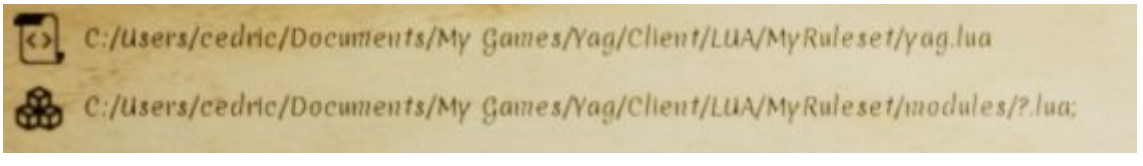
That directory must contain a file named ruleset.conf at its root:



That file contains 2 mandatory variables:

- LUA_FILE that contains the relative path of the file that Yag will call to run some LUA code. That path must be given relative to the directory <my documents>/My Games/Yag/Client/LUA/MyRuleSet.
- LUA_PATH that contains paths, also relative to <my documents>/My Games/Yag/Client/LUA/MyRuleSet and separated by **";"**. That variable is used to create the environment variable LUA_PATH (in which relative paths will be transformed to absolute paths by Yag) that tells LUA where to find modules.

The result of those 2 variables is displayed (and non modifiable) in the LUA panel:



If you have installed custom dice sets, the file ruleset.conf also contains 3 optional variables (CUSTOM_DICE_01, CUSTOM_DICE_02, CUSTOM_DICE_03) that let you specify the name of 3 customisable dice sets.
Watch out, the specified dice sets must be installed independently from the LUA directory. For more informations on customisable dice, you can watch the video on custom dice:

https://www.youtube.com/watch?v=ja6b62orsQE

Everytime a LUA mechanism is required, Yag will run the file LUA_FILE that must contain the functions from the API dedicated to the communication between Yag and LUA (see below).

By default, a directory default is created and overwritten each time Yag is started.
That directory contains a file ruleset.conf as well as a file yag.lua and a module default.lua, which define together the default Yag behavior.
That directory is an example that can be duplicated and renamed to serve as a starting point for any custom ruleset.

IMPORTANT: Please note that Yag currently doesn't share the LUA mechanisms between server and clients.
Every LUA script is run locally, every function is called locally.
This means that every player connected needs to install the LUA directory that will be used by their local copy of Yag.
This is generally as simple as unzipping an archive prepared by one of the players and setting its name in the LUA panel in Yag.

When required to run LUA, Yag proceeds as follow:

- always runs the script LUA_FILE once completely in order to identify and load LUA objects (functions…): if we put some code outside functions, it will be executed.
- optionally calls a LUA function from the Yag api, depending on the context

For example:

- The LUA button in the LUA panel only runs the script and calls no function. This can be used to check if the script is present, readable, if the modules are found, and, if the script does something, to use Yag as a LUA interpreter.
- When clicking the LUA button in the character sheets panel, Yag runs the script and then calls the Yag_CharacterSheetUpdate function, that must be provided by the player in the script.
- When clicking a dice button, Yag runs the script and calls the Yag_RollDiceButton function that must be provided by the player. If dice are rolled, after reading the results, Yag sends them to the player through the Yag_RollDiceCallback function, that must be provided by the player.

More about those 3 functions below.

Inside the LUA script, Yag is available through the object yag that contains various objects and functions.
For example:

- yag.LuaDir contains the full path for the <my documents>/My Games/Yag/Client/LUA directory.
- yag.PublicMessage() writes a public message to the journal

The structures

Some input and ouput of the functions have specific structures that are described here.

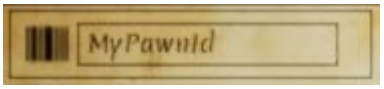
Character sheet (cs)

Character sheets in LUA are tables with the following structure:

```
{
  name = “my cs name”,
  maxhp = 27,
  init = “1d20+1”,
  lines =
  {
    id = { mod = true, extra = “whatever”, string1 = “type”, string2 = “name”, formula1 = “1d20”, formula2 = “1d8” },
    id = { mod = true, extra = “whatever”, string1 = “type”, string2 = “name”, formula1 = “1d20”, formula2 = “1d8” },
    id = { mod = true, extra = “whatever”, string1 = “type”, string2 = “name”, formula1 = “1d20”, formula2 = “1d8” },
    ...
  },
  ids = { 1 = “ID1”, 2 = “ID2”, … }
}
```

or more shortly:
{ name, maxhp, init, lines { id {mod, extra, string1, string2, formula1, formula2} ... }, ids }

maxhp is a number (max hit points of the character sheet)
mod is a boolean
All the others fields are strings.
ids is a table containing the ids of all the pawns attached to the character sheet. The id of a pawn is the one found in the pawn panel with the barcode icon:



Do not confuse line ids with pawn ids:

- cs.lines[“LineID”] = the cs line with id “LineID”
- cs.ids[1] = the id of the first pawn attached to the cs

Each id (also a string) should be unique. Please note that, to give a complete freedom to the user, Yag will not take care of the unicity of ids.
Objects (lines in a character sheet or pawns on the map) with identical ids may create chaotic behavior (disappearing of objects or incorrect updates).
For the lignes as well as the pawns, Yag creates a unique id using the letters “ID” followed by a sequential number.
For the character sheets lines this number is reset when the character sheet is empty of lines.
For the pawns, this number is reset when the map is emptied of pawns.

The fields correspond to the ones in the checked line here:

extra is a field that is not used by Yag, it is given for the player to use it for anything that must be used by LUA and stored in Yag: variables, conditions, lists of properties… It's just a free string.

The last table can contains as many lines as needed.

For example, if the character sheet hereabove is stored in the LUA variable cs:

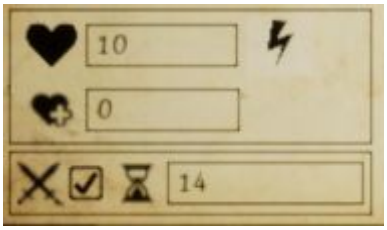
- cs.lines["w3"].formula1 will return "1d20-3"
- cs.lines["dex"].string1 will return "ability"
- cs.lines["w2"].extra will return "hammer"
- cs.lines["str"].mod will return false

Pawn sheet (ps)

A pawn sheet is a table containing some information about the pawns.
This is the structure of a ps:

```
{
 health = 10,
 initiative = 14,
 round = true
}
```

health and initiative are integers
round is a boolean telling if the pawn is part of the current action round
They correspond those fields in the pawn panel:



Dice results (dr)

Dice results must contain a lot of information: the original formula, the order, value, label and color of the rolled dice.
This is the structure of a complete dice roll "3d6 + 1d4 + 7":

```
{
 1 =
 {
  formula = "+bonus",
  color = "wood",
  result = { 1 = { value = 7, label = "7" } }
 },
 2 =
 {
  formula = "3d6",
  color = "white",
  result =
  { 2 = { value = 3, label = "three" }, 3 = { value = 6, label = "six" }, 1 = { value = 5, label = "five" } }
 },
 3 =
 {
  formula = "1d4",
  color = "black",
  result =
  { 1 = { value = 4, label = "four" } }
```

```
 }
}
```

or more shortly:
{ { formula, color, result { {value, label} … } }, ... }

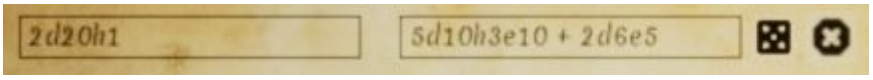
The key for each formula is a number
formula is a string
The key of each line { value, label, color } is the order of the dice in the formula: in “3d6”, a key “2” means that 3 was the second die rolled. In the example above, in the result for “3d6”, 5 was the first die, 3 the second one, and 6 the third one.

Note that LUA doesn’t ensure the order of its tables, so those keys are the only way to be sure about the order of the rolled dice.

dice roll origin (dro)

dro (dice roll origin) is a number from 1 to 3 telling the origin of the dice roll

- 1: rolled clicking a character sheet line button:
  - 

- 2: rolled clicking the dice panel button:
  - 

- 3: rolled clicking the init button:
  - 

The yag object (LUA to Yag)

From LUA we can access the yag object that will allow us to exchange with Yag.
It has the following properties.

yag.LuaDir

- The <my documents>/My Games/Yag/Client/LUA directory
- Useful if we distribute some LUA content and need a directory that will be recognized on every computer

yag.RollDice(f1, f2, ui, us)

- Ask Yag to roll the 2 formulas f1 and f2
- f1 and f2 are strings
- ui (user integer) is a free number that you can use to your convenience: it will be passed as it is to the Yag_RollDiceCallback function. You may use it to propagate dro from Yag_RollDiceButton to Yag_RollDiceCallback.

- us (user string) is a free string that you can use to your convenience: it will be passed as it is to the Yag_RollDiceCallback function.
- yag.RollDice(f1, f2) returns nothing
- After the dice have been rolled and the results read, Yag will call the callback function Yag_RollDiceCallback, see below for details

yag.ClearDice()

- Stop any current roll and delete all the dice
- yag.ClearDice() takes no argument and returns nothing

yag.PublicMessage(s)

- Write the string s in the journal
- All the players will see the message: use it for dice results
- s is a string
- yag.PublicMessage(s) returns nothing

yag.LocalMessage(s)

- Write the string s in the journal
- Only the caller will see the message: use it for personal logs or private dice rolls
- s is a string
- yag.LocalMessage(s) returns nothing

yag.GetPawnInfo(id)

- returns infos about the pawn identified by id
- id is a string that must be the id of a pawn
- yag.GetPawnInfo returns in this order:
    - a ps structure that is the ps of the pawn id
    - a cs structure that is the cs of the pawn id (it can be nil if the pawn is attached to no character sheet)
- ex: MyPawnPs, MyPawnCs = yag.GetPawnInfo("MyPawnId")

yag.UpdatePawn(ids, ps)

- Update all the pawns identified in the table ids with the ps structure.
- ids is a table of strings that must be ids of pawns: { "ID1", "ID24", "MyID", … }
- ps is a ps structure
- yag.UpdatePawn returns nothing
- Be careful: the same ps (with all its properties) will be applied to all the pawns in ids, so sometimes it is safer to update your pawns one by one using a table containing only one id:
    - yag.UpdatePawn({ "MyID" }, MyPs })

The LUA functions called by Yag (Yag to LUA)

When we need Yag to perform a LUA task, like updating a character sheet or customizing a dice result, we click buttons in the Yag UI.
Those buttons will call LUA functions in the player's LUA script.
Those functions must be provided by the player in their script, and are described here.

Yag_CharacterSheetUpdate(sp, cs)

- This function will be called when we modify a character sheet.
- sp (selected pawns) is a table containing the ids (string) of all the selected pawns.
- cs has the structure of a LUA character sheet (see the structures chapter above).
- This function takes a cs as an argument and returns a cs.
- This is the minimal function that must appear in a LUA script called by Yag (it does nothing as the cs is returned unmodified):

```
function Yag_CharacterSheetUpdate(sp, cs)
  return cs
end
```

When asked, Yag sends the currently selected character sheet as an argument to this function.
When getting the cs returned by the function, Yag will update the character sheet in the UI accordingly.

Yag_RollDiceButton(sp, cs, id, f1, f2, dro)

- This function will be called when we roll dice (by clicking a dice button in the UI).
- sp (selected pawns) is a table containing the ids (string) of all the selected pawns.
- cs has the structure of a LUA character sheet (see the structures chapter above)
- id is a string containing the unique id of the line to witch f1 and f2 belong
- f1 and f2 are two string containing the 2 formulas to be rolled
- dro is a number telling the origin of the dice roll (see the structures chapter above). You may propagate it to Yag_RollDiceCallback using the ui parameter of Yag_RollDice.
- Yag_RollDiceButton returns nothing but when the roll is over and the results have been read by Yag, the Yag_RollDiceCallback function will be called by Yag to allow the player to do whatever they want with the results of the rolls.
- This is the minimal function that must appear in a LUA script, it does nothing (dice will not even be rolled):

```
function Yag_RollDiceButton(sp, cs, id, f1, f2, dro)
end
```

Note that when called from a character sheet, f1 and f2 are not necessary as they can be retrieved from cs and id like so:

- f1 = cs.lines[id].formula1
- f2 = cs.lines[id].formula2

But sometimes there is no cs and no id (when a roll is asked directly from the dice panel for example), so they are provided in the input.

Also note that this function does nothing by default. If we wish so, we have to ask it to clear the dice dans roll them, with the yag.ClearDice and yag.RollDice functions.
It is hence possible to not use at all the dice from Yag. We just have to never call yag.RollDice and manage our dice with random numbers instead, with the rules we want.

Yag_RollDiceCallback(sp, cs, id, dr1, dr2, ui, us)

- This is the callback function after dice have been rolled either automatically (call by Yag to Yag_RollDiceButton(sp, cs, id, f1, f2)) or manually (call by the player to yag.RollDice(f1, f2)).
- sp (selected pawns) is a table containing the ids (string) of all the selected pawns.
- cs has the structure of a LUA character sheet (see the structures chapter above)
- id is a string containing the unique id of the line to witch f1 and f2 belong
- dr1 and dr2 have the structure of a dice result (see the structures chapter above)
- dr1 and dr2 are the respective results of the f1 and f2 rolls.
- ui (user integer) is the free integer ui passed by yag.RollDice.
- us (user string) is the free string us passed by yag.RollDice.
- Yag_RollDiceCallback(sp, cs, id, dr1, dr2) return nothing
- This is the minimal function that must appear in a LUA script, it does nothing:

```
function Yag_RollDiceCallback(sp, cs, id, dr1, dr2)
end
```