Ink-Fungus Gateway Template

Introduction

What is this?

The **Ink-Fungus Gateway** is (no surprise here) a gateway between Ink and Fungus. **Ink** is a scripting language for interactive fiction, developed by Inkle. **Fungus** is a visual storytelling and scripting tool for **Unity**.

The Ink-Fungus Gateway is available as a free package on the Unity Asset Store.

The best way to find out how the Gateway works is by checking the interactive documentation, provided as a playable Unity scene in the *InkFungus/Documentation* subfolder of this package. The interactive documentation is also playable in the browser on the author's itch.io page. This documentation is a useful reference for all the features of this tool. A good knowledge of Unity, Ink and Fungus is required to make games or other sorts of interactive text-and-pictures software using the Ink-Fungus Gateway.

Code-wise, the Gateway is mostly a set of `MonoBehaviour` components and new Fungus `Command` s. They manipulate the Ink engine so that it can plug into Fungus and provide ways to influence the Ink story flow from Fungus.

Who owns this?

The **Ink-Fungus Gateway** was created by Mauro Vanetti in 2020, during the pandemic, and released to the public domain.

It can be used freely, also for commercial games and other purposes. It can be modified and restributed, as long as credits to the original version are provided and the same free license applies. As an exception, no game of a racist, fascist or sexist character can be created using this tool or its derivatives. Why? Because screw nazis.

Ink is a free software created and maintained by Inkle Studios with the support of their open-source community.

Fungus is a free software originally created by Chris Gregan and currently maintained by a vibrant community of open-source developers.

Hopefully, you can become part of the community of developers of the Ink-Unity Gateway, too.

How can I set up a scene for the Gateway?

- Create an empty scene or open a scene you want to augment with Ink and Fungus.
- Import the Fungus package in the scene. *Lite* packages are OK.
- Import the Ink Unity Integration package in the scene.
- Find the *Tools > Fungus > Create* submenu. You have to create 3 objects from there:
    - First of all, create the *Ink-Fungus Gateway* in your scene. Click on *Tools > Fungus > Create > Ink-Fungus Gateway*.
    - Then, create a Fungus *SayDialog*. Click on *Tools > Fungus > Create > Say Dialog*.
    - Finally, create a Fungus *MenuDialog*. Click on *Tools > Fungus > Create > Menu Dialog*.
- You're also going to need an Ink script file. That's just a text file with a *.ink* extension. Put it somewhere in your project.

That's it.

You now have three key GameObjects in your scene:

- The `Ink-Fungus Gateway` with ist most important `Narrative Director` highly configurable component. It also includes a predefined Fungus Flowchart called `Gateway Flowchart` which is used by the default handling of narrative branchings.
- The `SayDialog` (a standard Fungus object) and all its childrens, grandchildren etc.
- The `MenuDialog` (another standard Fungus object) and all its childrens, grandchildren etc.

You can and should customise these objects, to modify their appearance and behaviour.

Basic features

Conversation

A *.ink* file containing the script of the interactive story is the core of any project based on the Ink-Fungus Gateway. Such a file will be written in the Ink language. Ink files are automatically compiled for usage by the Ink Unity Integration plugin provided by Inkle.

The Ink file asset used must be set in the `Narrative Director` component of the `Ink-Fungus Gateway` GameObject at *Basic Settings > Ink*.

Displaying text

Text lines in the selected Ink file are automatically displayed using the default or selected `SayDialog` GameObject. All Ink features, such as variations, variables interpolation etc. can be used to smartly generate text.

You don't always want the text to be displayed right away when the game starts. For this reason, the visual part of the game (Fungus-based) is going to initiate the narrative part (Ink-based). To this aim, the `Narrative Director` starts in pause mode.

The new Fungus command `Ink/Resume Narrative` is the way to go when you want Ink to start piloting your game. After that, the system will just process the Ink script and display its contents in the Fungus `SayDialog` .

Pausing and resuming the story

Ink scripts can contain tags. An Ink tag is a hash sign (*#*) followed by one or more words. Some tags are picked up by the `Narrative Director` and interpreted as special instructions.

`# pause` : the narrative will pause indefinitely *after* the line is displayed. Such indefinite pausing can only be ended by explicitly resuming the narrative flow.

`# pause N` : *after* the line is displayed, the narrative will pause for *N* seconds, where *N* can be any float number such as *10* or *3.5*.

`# wait` : the narrative will pause indefinitely *before* the line is displayed. Such indefinite waiting can only be ended by explicitly resuming the narrative flow.

`# wait N` : *before* the line is displayed, the narrative will pause for *N* seconds, where *N* can be any float number such as *10* or *3.5*.

The Fungus command `Ink/Resume Narrative` terminates the pause and resumes the story flow and it works both for indefinite and timed pauses.

When pausing/waiting or resuming happens, an *ink pause* or *ink resume* message is broadcast to all Fungus flowcharts, which can react accordingly.

The `# wait` tag was introduced for the common case in which the narrative flow needs some time to set up the view (e.g. to perform a fade animation) at the beginning of a new Ink knot or stitch. Using `# wait`, the event marking the new knot/stitch (*ink at*) will be triggered before the line is displayed. A working example can be found in the *Documentation\Examples* subfolder, it's called *Fading Views*.

Characters

In Fungus, every `SayDialog` contains parameters indicating where and how the text lines (*Story Text*) and the speaker's name (*Name Text*) are to be displayed. Check the Fungus documentation for details.

The Gateway introduces a conventional syntax to set up the `SayDialog` on the fly.

Text lines can be "plain" and contain no indication about who's speaking. No special syntax is required in this default case.

```
Once upon a time, there was a plain story line.
```

But a label with the speaker's name might be required instead. In order to indicate the speaker's name, it has to be put at the line's beginning, followed by a blank space and a quote sign (*"*).

```
Narrator "Let me tell you a nice story with a speaker's name in display.
```

The quote sign does not need to be "closed" at the end of the line (but it can, and the closing sign will not be displayed). Quotes can be used in the usual way within the text:

```
Narrator "Let me tell you a "nice" story with a speaker's name in display.
```

Fungus Characters can also be used in the conversation system. The only difference is that the speaker's label corresponds to the name of a GameObject containing a Fungus Character component. Every GameObject representing a character must be uniquely named for this syntax to work properly.

```
 Character1 "Let me introduce myself: I'm the first character.
Character2 "Glad to know you. I'm the second character of this story.
```

In this case, the `SayDialog` will be set up according to the Fungus Character's settings. The name displayed in the label will not be the GameObject name in the Unity Hierarchy, but, as expected, the value of property *Name Text* within the Character component. If a Character has the *Set Say Dialog* property pointing to another `SayDialog`, that one will be used instead.

If a character has a portrait, it will be displayed next to its name label in the `SayDialog` (this is standard Fungus). Characters can have multiple portraits, and one can be picked by writing a question mark followed by a portait tag:

```
 Character1?smiling "Let me introduce myself: I'm the first character!
Character2?shy "Hm, OK.
```

The portait tag is just any substring of the portait file name. The Gateway will look for a file containing it among the character's portaits. If no portrait tag is specified, the first portrait in the list will be used as default.

The colours of the name labels are defined character-wise, but a default colour for those speaker names that correspond to no character can also be configured. The default colour for character-less labels is specified in the *Default Character Color* property of the `Narrative Director`.

Branching narratives

Branchings with different options are defined in the Ink script according to the syntax of Ink. The Gateway assigns the options to the choices buttons of the Fungus `MenuDialog` through the predefined `Gateway Flowchart`. The appearance and behaviour of the menu can be modified by altering the `MenuDialog` as in vanilla Fungus.

The standard MenuDialog have a maximum of 6 option buttons. This means that the Ink script should not include more than 6 choices in any branching unless the `MenuDialog` is properly extended. In that case, the `Gateway Flowchart` must also be replaced or extended to include more than 6 options.

Hidden options don't count to the 6-option limit.

Hidden options

This is a feature introduced in version 1.2 of this Gateway.

If an option starts with *@*, it's called a hidden option. Hidden options are, rather predictably, hidden. They are not displayed as buttons with text.

Any kind of event can be associated to a hidden option (e.g., a click on a Fungus ClickableSprite) by using the new `Ink/Choose Hidden Option` Fungus command. This command takes one argument i.e. the hidden-option label without the leading *@* sign.

This is a huge advantage because a game story can be written in Ink, and then some of the textual choices can be turned into any kind of pixel hunt, drag-and-drop, puzzle solving interaction.

Thus, hidden options are a key feature to keep the text-and-choices layer well separated from the audio-video representation and interaction layer. A game with such layered architecture can be entirely tested in Ink before "wiring" it into its Unity project.

Visible options are not mandatory: a choice can include hidden options only, in which case no option button would be displayed and the narrative flow would not proceed until a hidden option is triggered.

Events

The Gateway sends a stream of messages to the Fungus flowcharts describing what's going on in the Ink-based narrative part.

Any event name always starts with *ink*, as in *ink pause* or *ink refresh n*. The *ink* prefix can be modified or removed altogether, though, in the `Narrative Director`'s settings.

Such messages may be intercepted by Fungus flowcharts in order to trigger the activation of blocks. In this way the visual side of the game can react to the story flow.

Pause, resume and stop events

When pausing, waiting or resuming happen, an *ink pause* or *ink resume* message is broadcast to all Fungus flowcharts. This can be useful to rearrange the screen and switch from the narrative mode to the visual mode, enable/disable controls etc.

When the story reaches a stop, an "ink stop" message is broadcast to all Fungus flowcharts. The stop event can mark the very end of the game or just a dead end of the narrative flow. The `Ink/Jump To` command can be used to get out of such dead ends. An *ink resume* message is also broadcast when, usually after a stop, a jump command is issued.

Loading and saving events

Loading and saving is described in details among the advanced features of the Gateway. Saving and loading happen in "slots" and slots can be told apart using their names.

At each save event, an *ink save* message is broadcast to all Fungus flowcharts. Another, more detailed message, is also broadcast, i.e. *ink save X* where *X* is replaced with the actual slot name.

There is also a lot of automatic saving going on in the *auto*, *precheckpoint* and *checkpoint* slots. These saves are not notified to Fungus.

The same happens with loading, *ink load* is broadcast followed by *ink load X* where *X* is replaced with the actual slot name.

There is also a special event for the outcome of the loading attempt: *ink load ok* or *ink load fail*.

(Don't call your saving slots *ok* or *fail* or *auto* or *precheckpoint* or *checkpoint* because that would mess everything up.)

These messages can be useful to display a floppy-disk icon, to prepare the scene to restart in another location etc.

Variable refresh events

Some variables are shared between the Fungus (visual) side and the Ink (narrative) side. The bidirectional updating is handled by the Gateway.

Every time a Fungus variable is automatically updated by the Gateway, two messages are broadcast to all affected Fungus flowcharts:

- a generic *ink refresh* message,
- and a more detailed *ink refresh X* where *X* is replaced with the variable name.

Flowcharts lacking any reference to the updated variable are not going to receive the message.

These messages can be useful to trigger a refresh behaviour, e.g. updating a score on screen or showing/hiding an item in the scene or inventory.

Story progress events

Whenever the narrative reaches a new knot or stitch, two messages are broadcast to all Fungus flowcharts:

- If it is a new knot, the message sent will be *ink at K* where *K* is the knot name.
- If it is a new stitch, the message sent will be *ink at K.S* where *K* is the knot name and *S* is the stitch name.

In all cases, a generic *ink at* message will also be sent.

These messages can be useful to change the scene according to the proceeding of the narrative, e.g. by fading in a new background picture or music track.

Generic events

Any unrecognized tag in the Ink script is echoed in the form of a broadcast message sent to all Fungus flowcharts.

To avoid interference with messages sent by other parts of the project, messages coming from Ink start with *ink*. For instance, a custom tag like this:

```
# fadeout 5
```

…will be translated as *ink fadeout 5*.

Variables

Global variables exist both in Ink and in Fungus, although there are some subtle differences between them:

- You can only have integers, floats, strings and Booleans in Ink. Other variable types that exist in Fungus cannot be synchronised as they are.
- You have lists in Ink, but not in Fungus. They undergo a special treatment.

The Ink-Fungus Gateway takes care of keeping in sync global variables that share the same name.

In order to do that, when a variable changes on the Fungus side, the Gateway needs to know, which can be done by executing the `Sync Variables` command. This new Fungus command can be found looking for `Ink/Sync Variables` in the command list.

When a variable changes on the Ink side, the Gateway detects it and the shared variable is automatically updated on the Fungus side. *ink refresh* messages are sent.

A working model for a Fungus flowchart to handle a shared global variable is the `Variable Processor` prefab included in the Ink-Fungus Gateway package. A `Variable Processor` can be created in your scene by clicking on *Tools > Fungus > Create > Variable Processor*. Please read the notes and comments in the `Variable Processor`'s flowchart to find out how to adapt it to your needs.

Jumps

If your interactive story is driven by the Ink script, you don't need any "jumps". The flow of your story can just follow the branching and weaving and tunnelling defined in your Ink script.

But you may want to have the visual (Unity/Fungus) side of your game drive the narrative sometimes. For example, you might make a game in which you freely walk around and sometimes you meet an NPC and a conversation begins. One of several possible ways to do that is using the new Fungus command `Ink/Jump To`.

The `Jump To` command is executed with a *K* or *K.S* parameter, where *K* is the destination knot and *S* is the optional destination stitch. When the command is executed, the story flow jumps to the new location within the Ink script. Variables (including those recording visited stitches and options) are preserved, but not the story flow stack.

Jumps are more useful in visual-driven games such as point-and-click adventures. Visual novels, on the opposite, are usually story-driven.

An example of safe usage of jumps is showcased in the interactive documentation. It uses Fungus blocks listening to *ink stop* and *ink resume* events as switches between story and visual controls.

Synchronised lists

Lists are a special kind of variables in Ink. Ink lists must be treated as a collection of Booleans (true/false) in Fungus, each Boolean variable representing whether a particular item is in the list (true) or not (false).

In order to establish a connection between Ink lists and Fungus Booleans, the global Booleans in Fungus are to be called *LIST__ITEM* where *LIST* is the Ink list name and *ITEM* is the Ink list item name.

As with ordinary variables, whenever a Boolean is modified on the Fungus side, the Gateway must be notified with an `Ink/Sync Variables` command so that the Ink side can be updated accordingly by including or excluding the corresponding item in the corresponding list.

Settings

The `Narrative Director` component of the `Ink-Fungus Gateway` GameObject has several settings that can be altered via the Inspector.

The settings are grouped in 4 areas in the `Narrative Director`'s parameters form.

Basic settings

This area is very straightforward. The only required value is the *Ink* field: you need to place your Ink script asset here.

The *Say Dialog* and *Menu Dialog* fields can be left empty: if there is only one instance of each required Fungus object, the Gateway will find it. If for some reason there are multiple Say Dialog and Menu Dialog objects, they can be specified there. Both values can also be replaced at runtime by executing the new Fungus commands called `Ink/Replace Say Dialog` and `Ink/Replace Menu Dialog`. This advanced feature can deliver interesting results, e.g. by differentiating conversation options from other sorts of menus (inventory, shops etc.).

The *Default Character Color* field specifies the text colour used for the speaker label when no corresponding Character is available. When a speaker name corresponds to an existing Fungus Character GameObject, their specific label colour is used.

Default Flags

Flags are binary (true/false) values that control some optional behaviours of the Gateway. Flags are set to their default values unless they are altered by the Ink script itself.

There are two ways to modify the value of a flag in the Ink script: permanent and temporary.

Permanent changes to a flag are applied after an *on FLAG* or *off FLAG* tag are processed (replace *FLAG* with the flag name, of course). Imagine this as permanently switching on or off that particular flag.

```
 Permanently switch off the echo feature. # off echo
Permanently switch on the auto feature. # on auto
```

Temporary changes last only for the time span of a conversation line (including the multiple-option menu if it's the line preceding a choice). Temporary changes to a flag are applied when a *yes FLAG* or *no FLAG* tag are processed (again: replace *FLAG* with the flag name).

```
 Temporarily switch off the hide feature. # no hide
Temporarily switch on the timer feature. # yes timer
```

To sum up: default flags are overriden by on/off tags, and both are temporarily overriden by yes/no tags.

The ***hide*** flag controls whether the last conversation line before a choice is kept in display or not when the options are shown. Ink does not allow for tags in choice options, therefore the *on/off/yes/no hide* tags are to be applied to the previous line.

The ***echo*** flag controls whether choices are treated as answers that are to be echoed back into the conversation. Notice that Ink was designed to treat them like that (as if the *echo* flag was always on) and has some special syntax to do that smartly. Again: Ink does not allow for tags in choice options, therefore the *on/off/yes/no echo* tags are to be applied to the previous line.

The ***auto*** flag controls whether the narrative proceeds automatically as soon as the text has been displayed, or not. Fungus conversation lines can be typed in gradually, therefore switching this flag on can make sense in some situations. It can also be useful for some dramatic effect. Remember that the speed, style and sound effects of typing can be tweaked in Fungus, altering the properties of the Say Dialog component.

The ***verbatim*** ("literally") flag controls the regular expression used to extract the optional speaker's name and portait from each Ink line. When the *verbatim* flag is on, Ink lines are rendered using only the Ink syntax, otherwise they also pass through the Gateway's parser.

The ***timer*** flag controls whether there is a maximum time to pick an option when the narrative branches. The timer is a feature of the Fungus conversation system, this flag only enables or disables it. When *timer* is on, the user has 5 seconds to pick an option, otherwise the first option in the list will be automatically selected. Once more: Ink does not allow for tags in choice options, therefore the "on/off/yes/no timer" tags are to be applied to the previous line. The countdown time can be changed in the *Advanced Settings* area of the Inspector and through the *timer* tag (not flag!).

Messages to Fungus for Ink Events

Some Ink events trigger the broadcasting of messages to the Fungus flowcharts. The Gateway handles these communications. In order to enable the adaptation of an Ink layer to a pre-existing Fungus game prototype, such messages are configurable.

First of all, every message has a universal prefix, *ink* followed by a blank space. This can be modified: it's the *Prefix For All Messages*. The blank space, if required, must also part of the prefix.

One can also remove the prefix altogether. In this case, watch out for colliding messages coming from Fungus itself or some Unity script.

The other properties just list all existing special messages sent by the Gateway and allow for them to be modified. Obviously, the Fungus Flowchart blocks should listen to the right messages.

Advanced settings

This area is only for advanced users.

The ***Regex*** parameter defines how incoming lines from the Ink engine are parsed by the Gateway to extract the optional character and portait parts. This is done through named groups in a regular expression. The main named group is called "text", there is an optional group named "character" and an even-more-optional group named "portrait". The regex is so complex because it needs to be lenient with the quote signs. This regex allow for quotes to be used in the actual text as long as they are properly opened and closed. There are some limitations in the characters available for the speaker label and the portrait tag, which can be a problem for some. As long as there are the grouped names "text", "character" and "portrait", an alternative regex can be used instead of the default one. The default regex is:

```
/^(?<character>[\w\-]*)(\?(?<portrait>[\w\-]+))? "(?<text>(("[^"]*")|([^"]+))+?)"?[ ]*$/
```

The ***Gateway Flowchart*** parameter defines the Fungus Flowchart used to handle the player's choice at branching points of the story flow. Basically, what the Gateway does is associating each option with the corresponding block inside a special flowchart. According to the option picked by the player, the new command `Ink/Choose Option` is called with a 0-based integer argument. If for any reason this behaviour were to be delegated to another flowchart, that can be specified here.

The ***Choice Time*** parameter defines the default duration (in seconds) of the countdown. The countdown is switched on by the *timer* flag and limits the time available for the player to pick an option when a `MenuDialog` is shown. This is the default value but the choice time can also be modified in the Ink Script like this:

```
From now on, we get one minute every time we have to choose something. # timer 60
```

The ***Hidden Option Prefix*** parameter defines the prefix used to mark hidden options, i.e. options that are not to be shown in the `MenuDialog`. Changing this to any character or sequence of characters used in Ink's syntax would break the Gateway, so please be careful.

The ***Enable Save And Load System*** parameter enables the system to save and load the story state. Many projects don't need the save/load system and can safely disable it for performance reasons, particulary to prevent frequent access to the local disk. This option is checked by default. Furthermore, if you have localization in your project, your choice will be overriden to always be enabled; the reason for this is that when you switch language, your latest autosaved state is restored.

Save and load

In most projects heavily relying on Fungus, implementing a save-and-load system in parallel with Ink is going to be a nightmare. The Gateway provides some tricks to help the process but please consider this feature as almost experimental and prone to failure.

Saving

The Ink-Fungus Gateway allows for saving the Ink state in two different modes: *snapshot* and *checkpoint*.

By executing the new command `Ink/Save Snapshot` (which can be done from a Fungus Flowchart block) you save the current state at the moment when the current conversation line was displayed. Subsequent changes in variables (done through Fungus) are not going to be saved.

By invoking another new command, `Ink/Save Checkpoint`, you save the story state at the beginning of the current knot and stitch.

This follows two different styles of saving used in many video games, even though it doesn't really allow for moment-to-moment save. Obviously you can restrict the saving even more than that, by letting that happen only at some special milestones of your game, like chapters.

Both save styles ask for a slot name. This is going to be used to form the file names for the save files (each save is several files).

Notice that there is a lot of automatic saving going on, using slots *auto*, *precheckpoint* and *checkpoint*. Don't call your slots like that.

The default slot name for manual saves is, predictably, *manual*. If you just need one slot, use that one.

Loading

To load what you saved in a slot, execute `Ink/Load` with the slot name as an argument.

Messages are broadcast at save and load events.

Remember that everything Fungus (and even worse, everything Unity out of Fungus) is not going to be magically stored in your Ink save files. The Gateway can tell Ink to save the state of the narrative but not the state of the visuals, including the non-global variables etc. This has to be managed separately and simultaneously in Unity/Fungus.

Localization

Localization is complicated. Fungus has its own way of translating text, which is fine but it's not what we're using in the Gateway.

The best way to localize an Ink script is to write another Ink script with the same knots-and-stitches structure, the same variables, the same logic. The Gateway helps in switching between "twin" Ink scripts in different language, by means of the `Alternate Language Narrative Director` component. This is a `MonoBehaviour` script provided with this package.

In interactive narrative, as in narrative in general, there cannot always be a 1-to-1 relationship between translated sentences. For this reason, the checkpoint logic is applied to language switching in the Gateway. This means that if the player switches to a new language, they have to start from the last checkpoint. That's why we have the *precheckpoint* save slot: that's where we rewind our story back to, when we switch language.

In order to add extra languages to your game, you need to attach one instance of the `Alternate Language Narrative Director` component per extra language. The `Alternate Language Narrative Director` component has two properties: a language tag and the corresponding Ink script. Be careful in keeping the translated Ink script identical to the original version as far as knots, stitches, variables and logic are concerned.

The new command `Ink/Switch Language` must be executed with a language tag argument in order to switch to the corresponding translation. When the `Ink/Switch Language` is executed with an empty argument, it restores the original language.

Also, consider other paths. For example, you may force the player to pick a language at the beginning and then stick with it. This solution would only require one multi-lingual Ink file (but every knot and stitch must be duplicated).