# **LOGIKA**

# **What is Logika?**

Logika (Logic in Esperanto) is a FSM ([finite state machine](https://en.wikipedia.org/wiki/Finite-state_machine)) implementation for Ren’Py, FSM have a draw (graphic) that is a representation of the logic and the flow of process (label for Ren’Py).

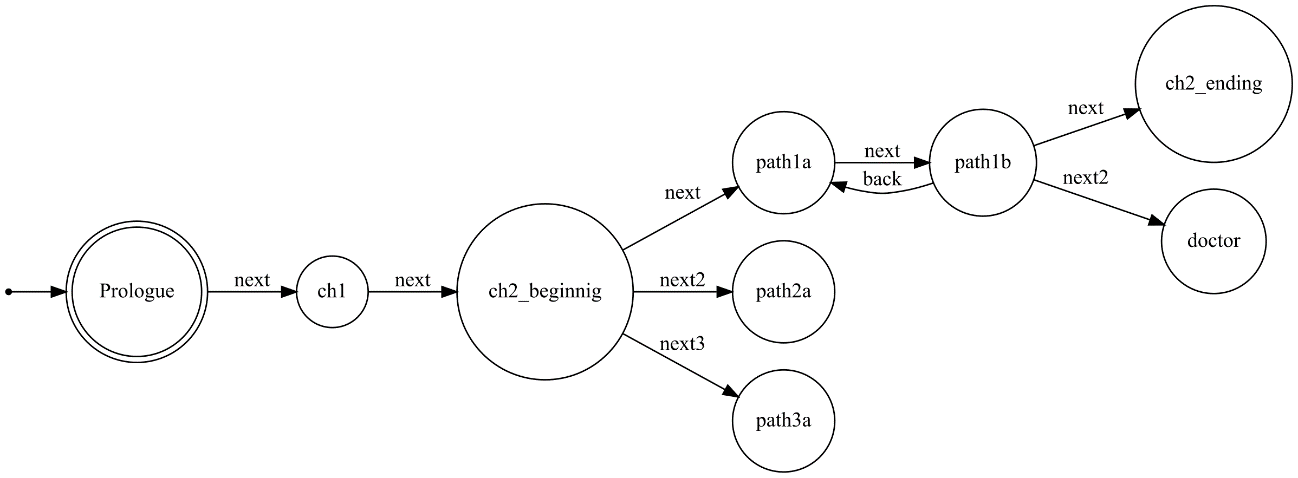


Figure 1 Flowchart example

We have three element of interest the Start label, scene (Label in renpy) and transition condition this will named Logika elements.

|  |  |
| --- | --- |
| **Name** | **Graph representation** |
| Start |  |
| scene |  |
| Transition condition |  |

Table 1 Elements of Logika element

The “scene” or “start” have a representation in Ren’Py

|  |  |
| --- | --- |
|  | Label ch2\_beginning:  A “blab la bla”  …  …  If condition\_inner1: Return “next”  If condition\_inner2: Return “next2”  If condition\_inner3: Return “next3” |

Table 2Scene in Logika Unit is a Ren’Py label

The Logika Unit have a representation in logic.

|  |  |
| --- | --- |
|  | IF path1a SEND next GOTO path1b IF path1b SEND back GOTO path1a |
|  | START Prologue |
|  | START Prologue IF Prologue SEND next GOTO ch1 IF ch1 SEND next GOTO ch2\_beginning IF ch2\_beginning SEND next GOTO path1a IF ch2\_beginning SEND next2 GOTO path2a IF ch2\_beginning SEND next3 GOTO path3a |

Whit the logic, we can translate to renpy(python) logic using an aplication or doing manual.

IF path1a SEND next OR next2 GOTO path1b

This translation is like this.

If scene==”path1a” and transition in [“next”,”next2”]: scene=”path1b”

Logika includes an application to perform this transfer faster

Imagen que contiene captura de pantalla

Descripción generada automáticamente

# **Why use this?**

Well, when your history/game have very complex paths, is more easy only change the logic, with that you can concentrate in the writing process. For example you need to add a new scene between two scenes, that need to check the code and validate that don’t break the code, but with Logika (FSM) you only change the logic add a new line and change one. The logic before change:

IF scene\_old1 SEND next GOTO scene\_old2  
IF scene\_old2 SEND next GOTO scene\_old3

The logic after change:

IF scene\_old1 SEND next GOTO scene\_new1  
IF scene\_new1 SEND next GOTO scene\_old2  
IF scene\_old2 SEND next GOTO scene\_old3

Think in Logika like the oldest choose your adventure book, like a book you change page and don’t need to think what will show, this validate the logic.

# **How to use?**

Create the class with the application or do it manual, then write this in your code (script.rpy).

|  |
| --- |
| Label start: $ logic\_game=FSM\_logic()  $ state=logic\_game.logic("")  call expression state  while (\_return!="gracias"): #key phrase to finish  $ state=logic\_game.logic(\_return)  call expression state  # $ print logic\_game.read\_logger()  return |

Table 3 script.rpy code

## **The Class**

|  |
| --- |
| init python:  import math  import pickle  class FSM\_logic(object):  '''Finite State Machines logic for game'''  def \_\_init\_\_(self, label\_0="Prologue",logger=False):  self.state=label\_0  self.logika = logger  self.log\_FSM=[]  def logger(self,data):  f=open("debug.img","wb")  pickle.dump(data, f, pickle.HIGHEST\_PROTOCOL)  f.close()  return 0  def read\_logger(self):  s=file("debug.img","rb")  value=pickle.load(s)  s.close()  return value  def logic(self,command):  old\_state=self.state  ####### **put logic here**  if self.state=="Prologue" and command=="next": self.state="ch1"  elif self.state=="ch1" and command=="next": self.state="version\_end"  #######  self.log\_FSM.append(self.state)  if self.logika:  self.logger(self.log\_FSM)  if renpy.has\_label(self.state):  return self.state  else:  renpy.log("Warning: Something happened there is no label: {}  / Class FSM\_logic".format(self.state))  renpy.notify("Something happened there is no label:  {}".format(self.state))  #puedo configurar para que retroceda un paso  return old\_state  #return "error404" |

This class have this structure, you only need to use logic(<transition condition>) and receive the next scene (label name), logger and read\_logger is for have a record of the path, this create a file “debug.img”. label\_0 is the start point in FSM.

|  |
| --- |
| **FSM\_logic** |
| self.state=label\_0 self.logika = logger self.log\_FSM=[] |
| Logic(command)  read\_logger()  logger(data) |