MAPA ONÍRICO

ONIRISCHE KAART

EXPLORING DREAMS TROUGHT DIGITAL NARRATIVES

TWINE BOOKLET

UNO

DREAM S
DE- FRAG -
MENT

’

WHAT'S TWINE?

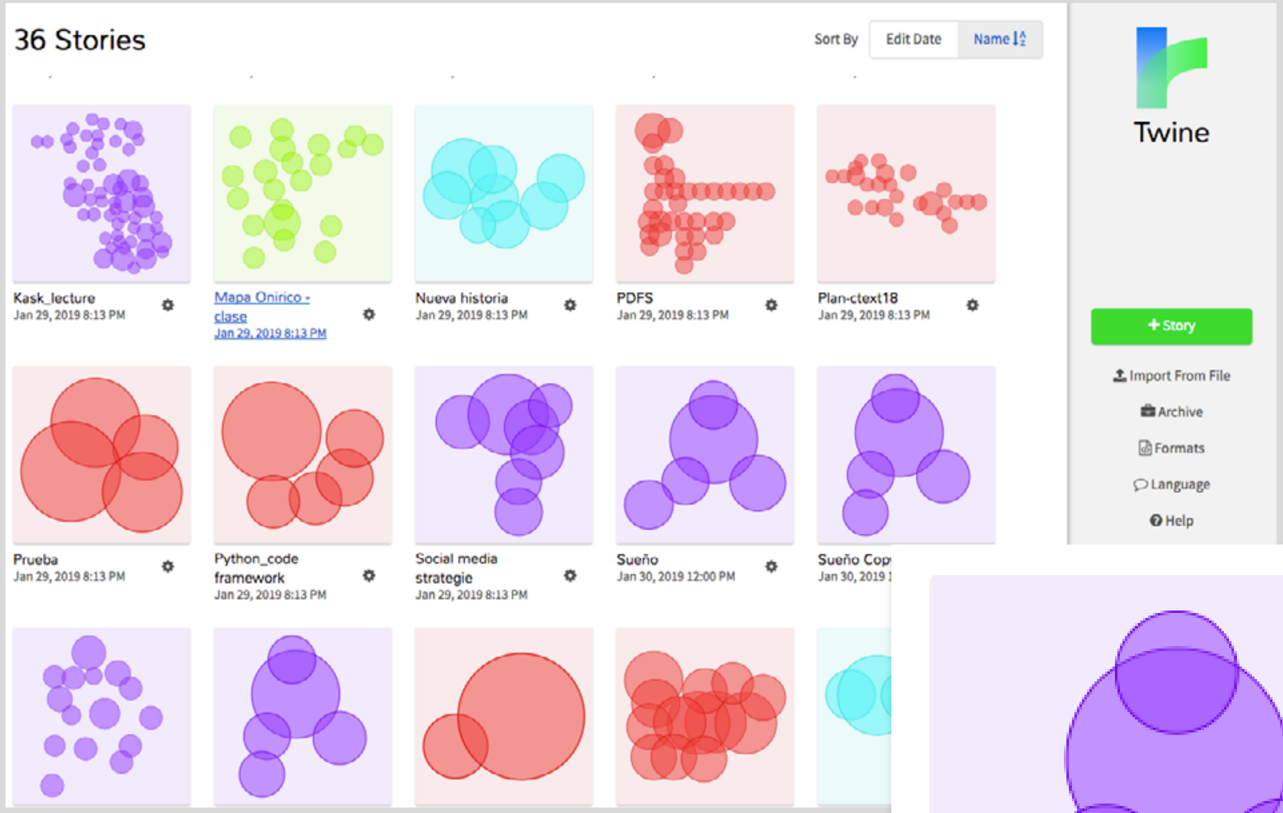
- Free open source-tool
- Hyper text and games
- Visual structure of the hypertext
- Twine 2 is web base api written in HTML5 and JS
- Twine 2 is also a desk app

FORMATS

- ⇢ *Harlowe*
- Snowman
- SugarCube

STORY

-IDFD



36 Stories

Sort By Edit Date Name

Kask_lecture Jan 29, 2019 8:13 PM

Mapa Onirico - clase Jan 29, 2019 8:13 PM

Nueva historia Jan 29, 2019 8:13 PM

PDFS Jan 29, 2019 8:13 PM

Plan-ctext18 Jan 29, 2019 8:13 PM

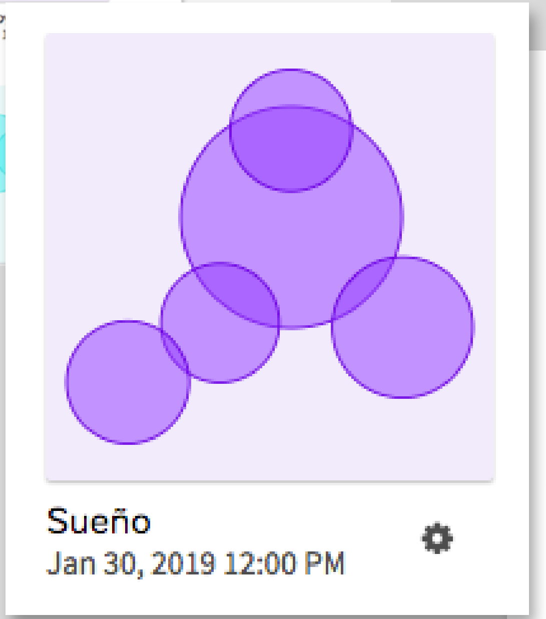
Prueba Jan 29, 2019 8:13 PM

Python_code framework Jan 29, 2019 8:13 PM

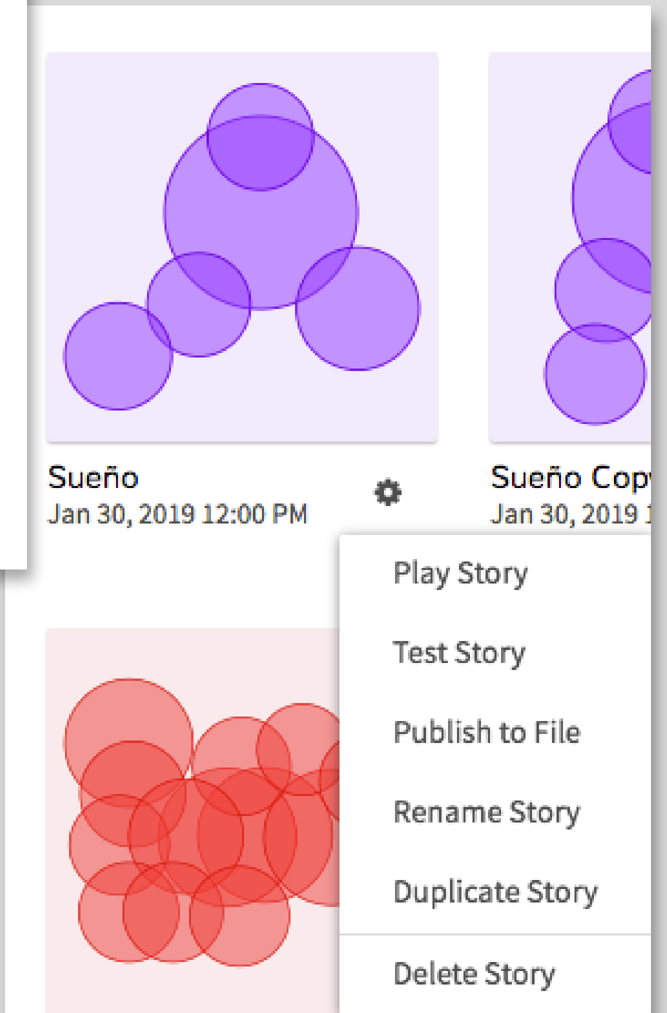
Social media strategie Jan 29, 2019 8:13 PM

Sueño Jan 30, 2019 12:00 PM

Sueño Cop Jan 30, 2019



Sueño
Jan 30, 2019 12:00 PM



Sueño
Jan 30, 2019 12:00 PM

Sueño Cop
Jan 30, 2019

A) NUMER OF STORIES

B) SORT DATE/NAME

Arrange the stories in alphabetical order or last time modification

MAIN MENU

C) STORY

Create a new story

D) IMPORT

Import a published story or a Twine archive.

E) ARCHIVE

Save all stories to a Twine archive file

F) FORMAT

Choose a format for your story: Harlowe, Snowman or SugarCube

G) LANGUAGE

Choose the interface language

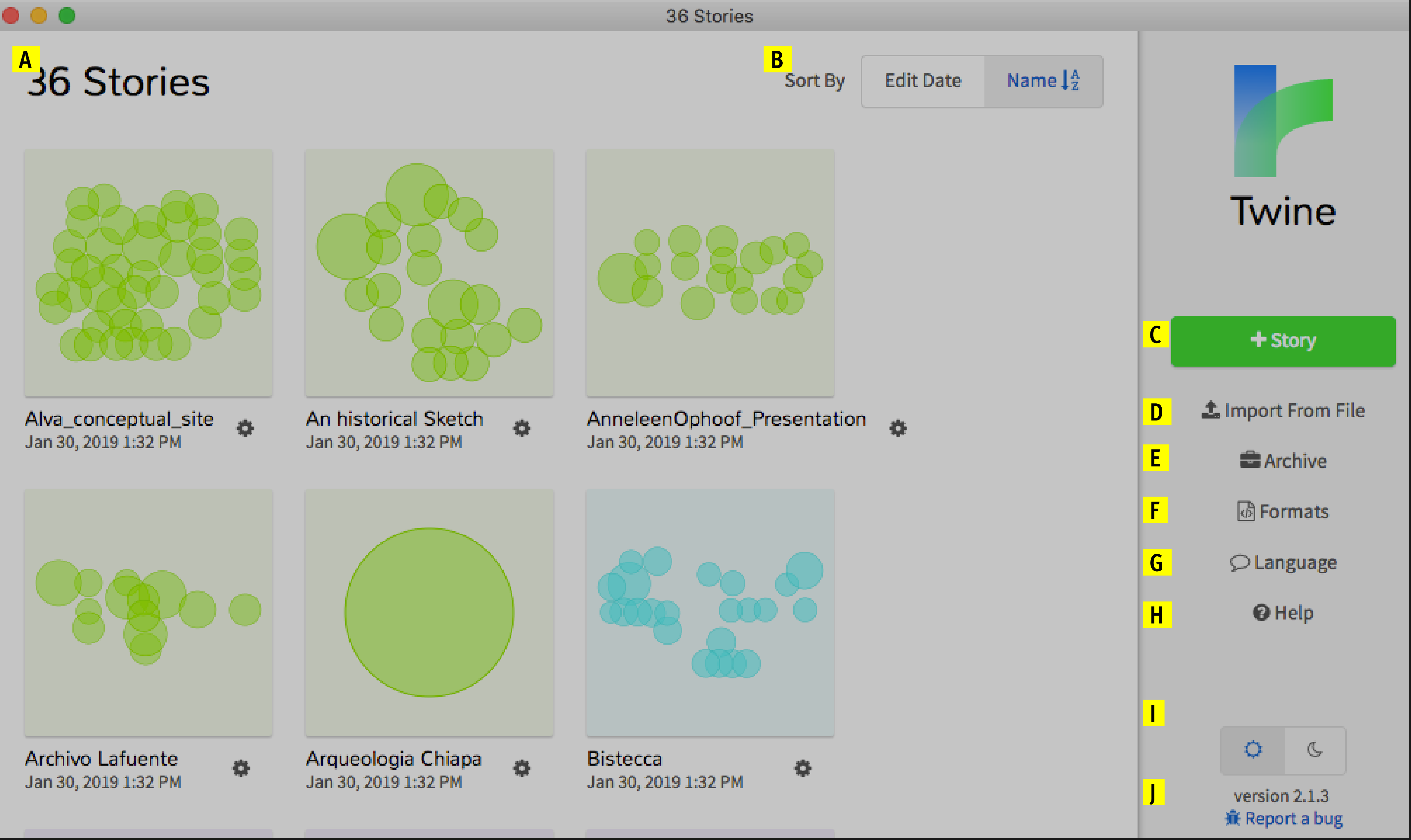
H) HELP

Web support base in Twine documentation

I) VISUAL MODE

Toogle between night and light visual mode

J) TWINE VERSION



36 Stories

A 36 Stories

B Sort By Edit Date Name

Alva_conceptual_site
Jan 30, 2019 1:32 PM

An historical Sketch
Jan 30, 2019 1:32 PM

AnneleenOphoof_Presentation
Jan 30, 2019 1:32 PM

Archivo Lafuente
Jan 30, 2019 1:32 PM

Arqueologia Chiapa
Jan 30, 2019 1:32 PM

Bistecca
Jan 30, 2019 1:32 PM

Twine

C + Story

D Import From File

E Archive

F Formats

G Language

H Help

I

J version 2.1.3
Report a bug

UI MAIN MENU

A)HOME
Go to Storie list

B) STORY NAME

C)OPTIONAL MENU
Toogle optional menu

D)EDIT JAVASCRIPT
Insert and edit Js

E)EDIT CSS
Insert and edit CSS

F)CHANGE FORMAT
Select story format : Harlowe, Snowman & SugarCube

G) CHANGE STORY NAME
Edit story name

H) SNAP TO GRID
Snap passage to the user interface grid

I) STORY STADISTICS
Show stadistics and the IFID name of the story

J) VIEW OF PROOFING
View all the story text

k) PUBLISH FILE
Publish files

L) QUICK FIND
Search for a term in the story

M) FIND MENU
Find and replace across the enire story

N) ZOOM STRUCTURE
Show the story structure

O) ZOOM PASSAGES
Show passage titles

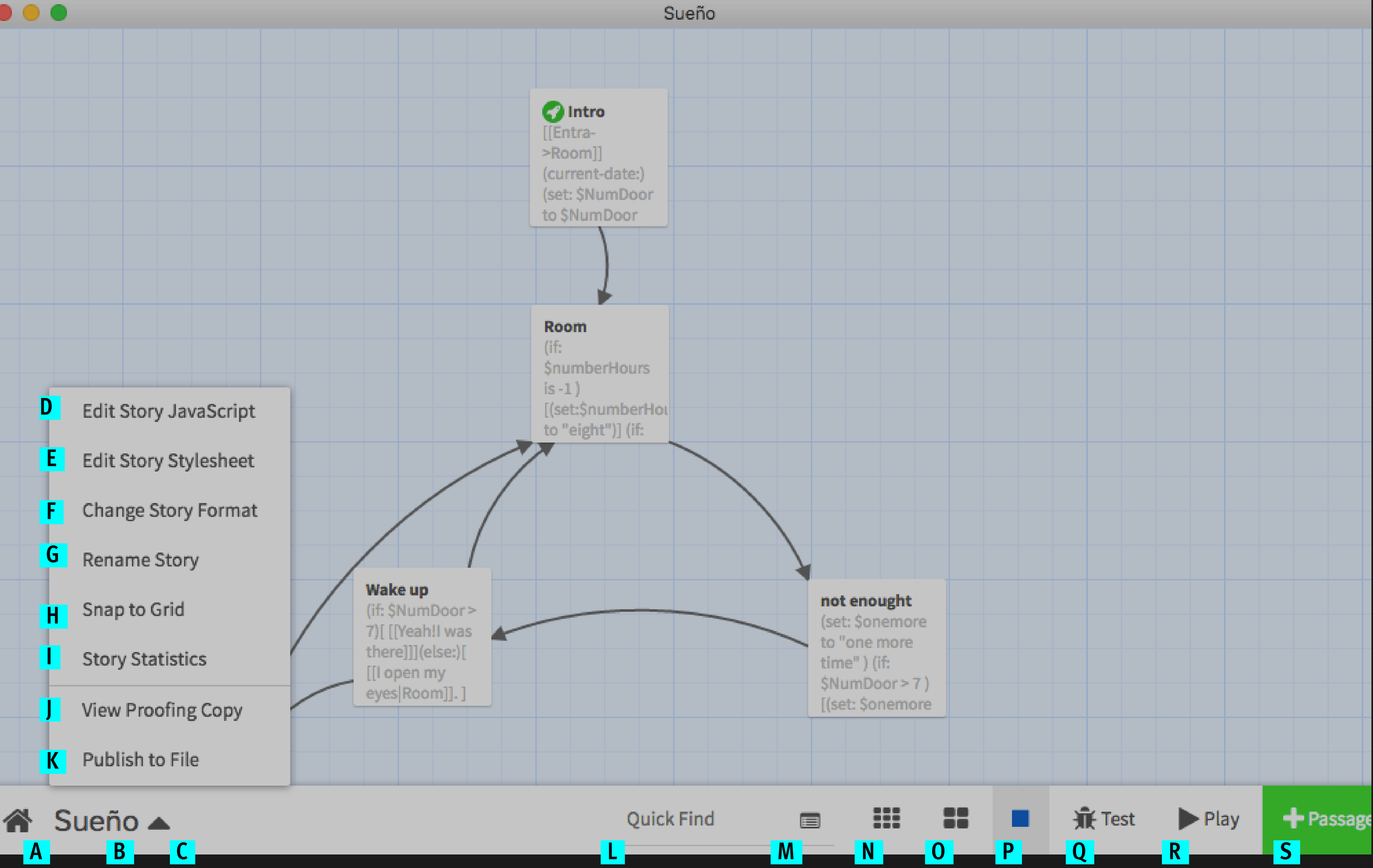
P) ZOOM
Show passage titles and excerpts

Q) TEST MODE
Play the story in debugging mode

Q) PLAY
Play the story

Q) PASSAGE
Create new passage

UI STORY

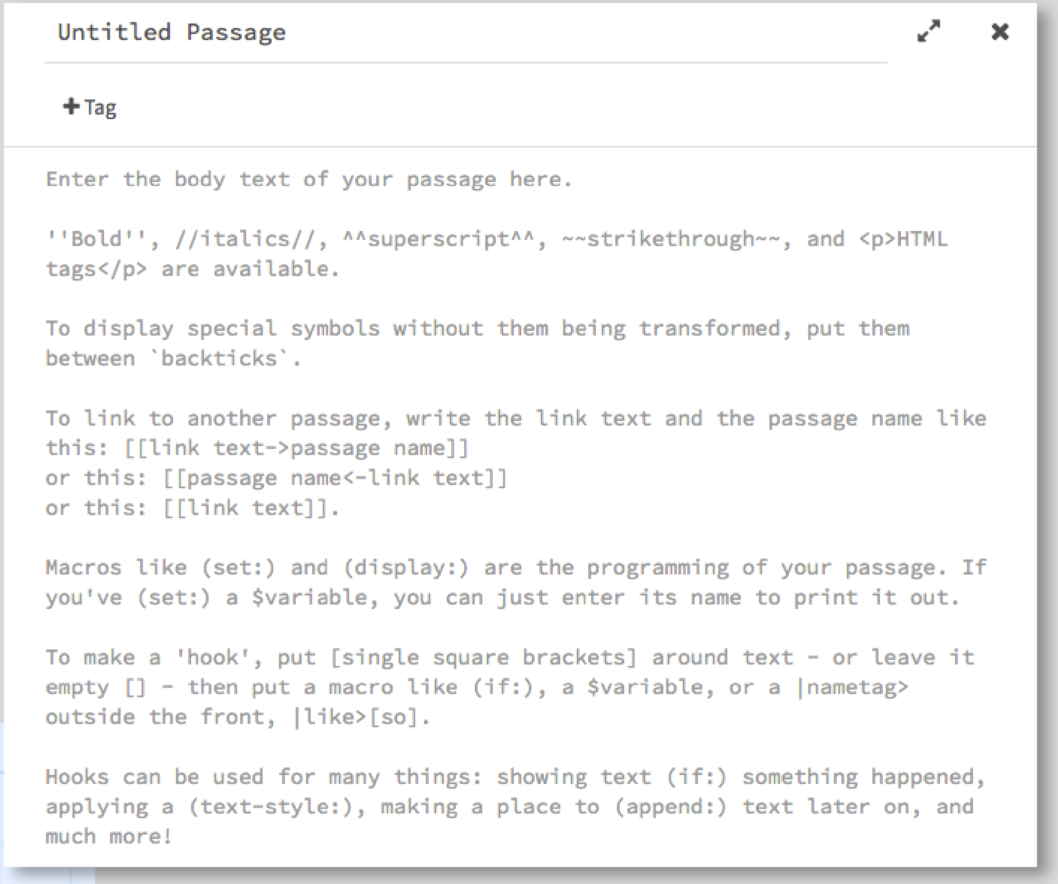

Sueño
Intro
[[Entra->Room]]
(current-date:)
(set: $NumDoor
to $NumDoor
Room
(if:
$numberHours
is -1)
[(set:$numberHou
to "eight")] (if:
Wake up
(if: $NumDoor >
7)[[[Yeah!I was
there]]](else:)[
[[I open my
eyes|Room]].]
not enought
(set: $onemore
to "one more
time") (if:
$NumDoor > 7)
[(set: $onemore
D
Edit Story JavaScript
E
Edit Story Stylesheet
F
Change Story Format
G
Rename Story
H
Snap to Grid
I
Story Statistics
J
View Proofing Copy
K
Publish to File
Sueño
Quick Find
Test
Play
Passage
A
B
C
L
M
N
O
P
Q
R
S

UI STORY

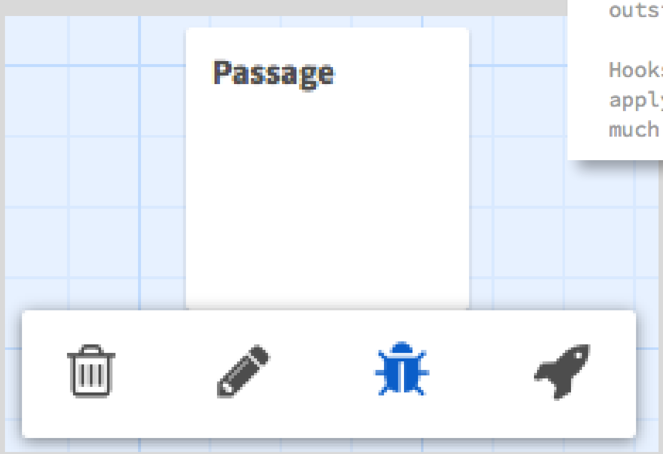
Passage

Basic HTML

ex: <h1>, <p>,<br>, <div>, <i>,


Untitled Passage

```
Enter the body text of your passage here.

''Bold'', //italics//, ^^superscript^^, ~~strikethrough~~, and <p>HTML
tags</p> are available.

To display special symbols without them being transformed, put them
between `backticks`.

To link to another passage, write the link text and the passage name like
this: [[link text->passage name]]
or this: [[passage name<-link text]]
or this: [[link text]].

Macros like (set:) and (display:) are the programming of your passage. If
you've (set:) a $variable, you can just enter its name to print it out.

To make a 'hook', put [single square brackets] around text - or leave it
empty [] - then put a macro like (if:), a $variable, or a |nametag>
outside the front, |like>[so].

Hooks can be used for many things: showing text (if:) something happened,
applying a (text-style:), making a place to (append:) text later on, and
much more!
```



A B C D

A) ERASE

B) EDIT STORY

C) PLAY DEBUG VIEW

D) SET STARTING POINT

LINK PASSAGE SINTAX

- [[name of the passage]]
- [[name of the passage | Display text]]

- [[text to display -> Name of the passage]]
- [[Name of the passage <- text to display]]

VARIABLE

A *variable* is a place to store data.
It is a symbol that represent a value

$X = 5$

$y = 3$

$X + y = Z$

$Z = 8$

There are global variables

$VariablePeople SINTAX

- **Recomend to Set** at the starting of your game.
- You can modify their value in any passage.
- They behave globally
- You can not named using numbers. “Ex. $573 ”

And local variable

_OcacionalPeople SINTAX

These variable only exist during the current passage where you place them

TYPE OF DATA

Boolean: false or true

String : Secuences of characters (text)

Ex: "hola", "hey", "dos"

Integer: Numerical value.
Any natural number.

Ex:2, 5 ,-1,

MACROS

It is a piece of pre-defined code inserted in your *passage*.

A macro is an instruction. There are diverse *macros*, each of them has it’s own requirements .
EX. “(Set:)” ”(If:)” “(Display:)”

SET A VARIABLE

- (set: $NameVariable to 5) **Sintax** Here the variable is an integer

- (set: $NameVariable to “five”) Here the variable is a string

- (set: $NameVariable to “five”+ $otherVariable + “hands”)

“Set” allows you to add diverse variables, you can use operators as +

- (set: $NameVariable to “5”, $NameVariable2 to “cinco”)

“Set” allows you to synthesize work by condensing diverse variables in one stament
. Use comma to separate each of them

IF

CONDITIONAL STAMENT *«IF»*

BLOCK TO PERFORM

```
(If: $NameVariable is "5" )[ (set: $NameVariableToString to "cinco") ]
```

THE RESULT OF THIS WILL CHANGE INTEGER “5” THE TO STRING FIVE OR CINCO IN SPANISH

IMAGE

Absolute path: absolute url

```
<img src="https://twinery.org/homepage/img/logo.svg"
width="256" height="256">
```

Relative path: relative to the story file

```
<img src="../img/logo.svg" width="256" height="256">
```

HTML ELEMENT « IMG » COMPONENTS

• src (Source of the image). *Mandatory

• alt (Alternative text .Descriptive)

• width

• height

NOTE : Check always your image source path

This could be Absolute, fixed to a URL adress or relative to twine html file

White space

You may have noticed that every time we use a macro, some weird extra space appears in the rendered output.

One way to fix it is by using {} as a wrapper of those macros.

```
{

(set: $Jeng to "just enought")

(set: $arrayUwere to (a:"<br>You were there", "You were ab-
stent") )

}
```

Another way to do this is with the \ character (backslash). If you put this character at the end of a line, it instructs Twine to not display a new line in the rendered output:

```
This line\

and this line

\and this line, are actually just one line.
```

Hooks

A command that allows the author to give a hook a computed tag name.

[some random text]

|name>[some random text with name]

[some random text with name 2]<name2|

OJO
CÓMO
PALABRA

REFERENCES

‹Sugar cube reference›

▸ Parish, Allison “A Quick Twine (2.2+) Tutorial” [online] Available in http://catn.decontextualize.com/twine/

‹Harlowe reference›

▸ Cox, Dan [video online] Available in https://www.youtube.com/playlist?list=PLlXuD3kyVEr5tlic4SRe6ZG-R9OyS1T4d

▸ Vegetarian Zombie "Introduction to Twine" [video online] Available in https://www.youtube.com/watch?v=1jukyU4EK2M

▸ "Twine cookbook" [online] Available in https://twinery.org/cookbook/

▸ "Harlowe 2.0.1" [online] Available in https://twine2.neocities.org/