LoveStory

A Visual Novel Library for LovePotion on the Nintendo Switch

1. Introduction and License

LoveStory is a VisualNovel library for LovePotion on the Nintendo Switch. LoveStory's purpose is to simplify the creation of VisualNovels as well as the integration of a simple to use DialogueSystem for story driven games for Nintendo Switch.

LoveStory may only be used with LovePotion for Nintendo Switch. LovePotion is a Nintendo Switch port of the LUA based game engine Love2D by TurtleP.

LoveStory runs under the **MIT-License**:
This means LoveStory is completely OpenSource. You are allowed to use it in all kind of projects and you are free to modify all contained files, but I will not be responsible for anything that is produced or used in relation with this library. You can find the MIT-License in the next chapter.

I would be very happy if you mention me somewhere in your game's credits if you used this library ☺

LoveStory was initially created by ***Shrike*** in 2018, so was this documentation.

More Information about **LovePotion** for Nintendo Switch:

https://github.com/TurtleP/LovePotion/releases/tag/switch-1.0.0

https://github.com/TurtleP/LovePotion/wiki

More Information about **Love2D**:

https://love2d.org/

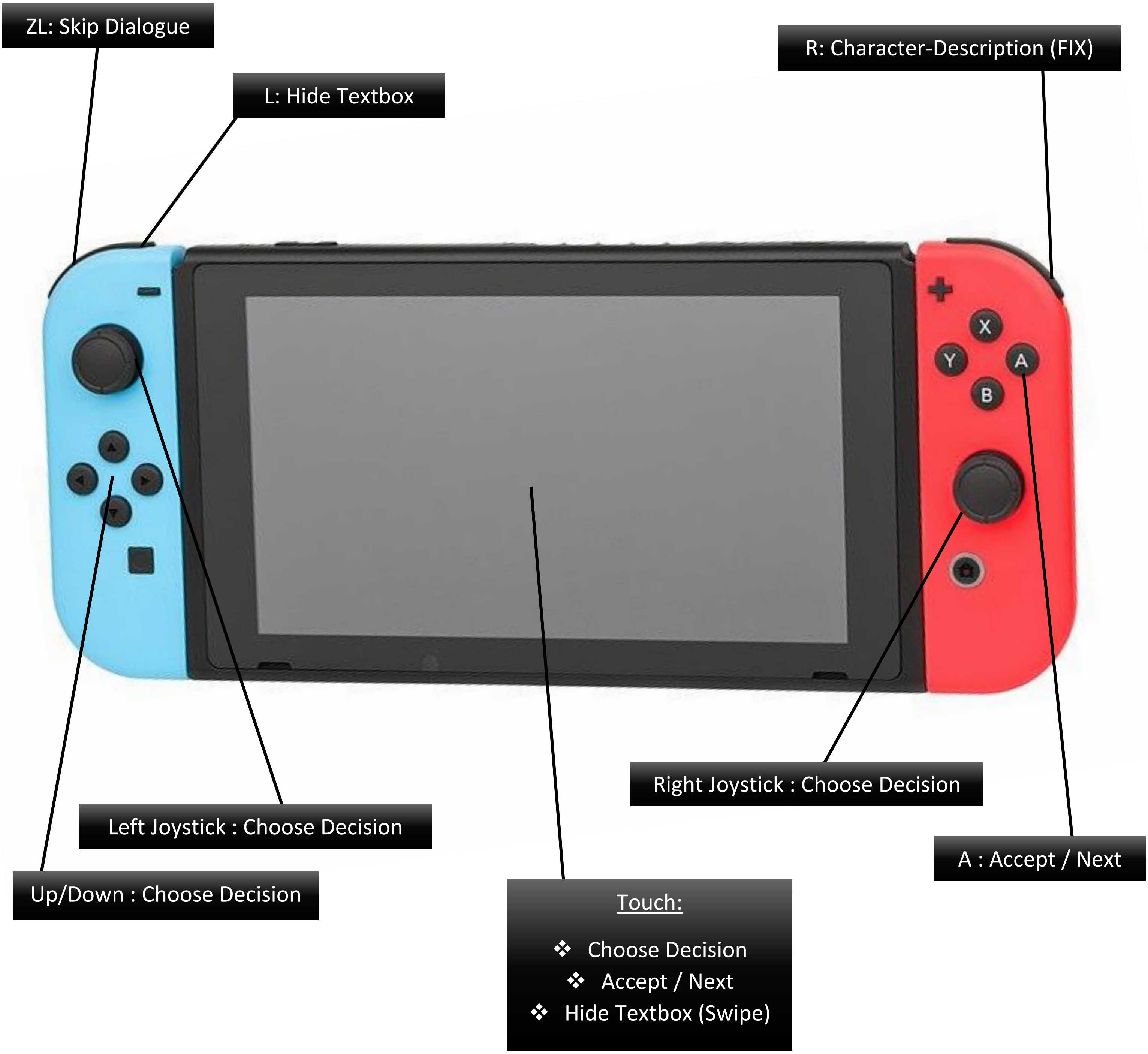
https://love2d.org/wiki/Main_Page

1.1 MIT-License

2. Features

- **Easy to use** dialogue implementation for the creation of VisualNovels or other story-driven games
- **No fancy programming skills needed** to create VisualNovels with the help of this library
- Changeable character pictures for **emotions** and **clothing-sets**
- **Decision making for complex story-lines and multiple endings**
- Two characters can be shown the same time
- **Complete touch-support** for easy game control
- Dialogue boxes can be hidden with touch-swipe or joycon-input
- **Sound and music support:** You are able to create full dubbed versions of your in-game conversations
- Easy managing of multiple characters thanks to character-classes
- **Special character-screen** when holding the **switch in portrait-mode**, with a big picture and more character information shown
- You are able to show a background image with or without your characters in front and easily change it anytime
- **Skip all the current dialogue** with only holding down the LZ-key (shoulder-button)
- **Future Updates**: If the community has wishes for future improvements or features I will try to implement them in one of the next versions

3. Game Controls



ZL: Skip Dialogue

L: Hide Textbox

R: Character-Description (FIX)

Right Joystick : Choose Decision

Left Joystick : Choose Decision

A : Accept / Next

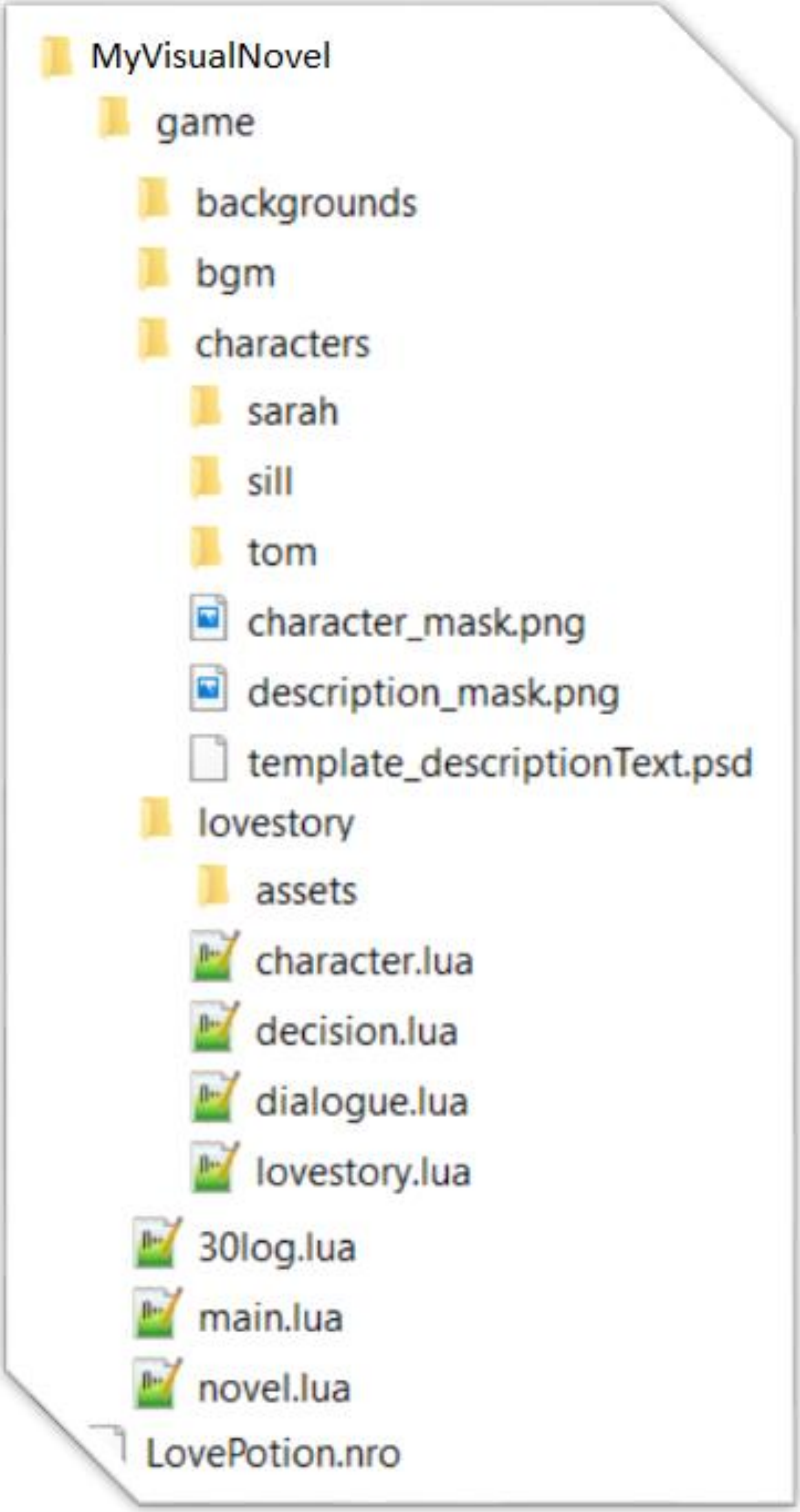
Up/Down : Choose Decision

Touch:

- Choose Decision
- Accept / Next
- Hide Textbox (Swipe)

4. Folder-Structure and Data-Specifications

In the picture below you can see the given folder-structure:



- MyVisualNovel
  - game
    - backgrounds
    - bgm
    - characters
      - sarah
      - sill
      - tom
      - character_mask.png
      - description_mask.png
      - template_descriptionText.psd
    - lovestory
      - assets
      - character.lua
      - decision.lua
      - dialogue.lua
      - lovestory.lua
    - 30log.lua
    - main.lua
    - novel.lua
  - LovePotion.nro

- *game/backgrounds*
    - Insert background images here.
      Name: <NUMBER>.png *(example: 0.png)*
      Format: 1280x720
      File: PNG
      You can use *game/backgrounds/bg_mask.png* as template to adjust your own background to the specification.

- *game/bgm*
    - Insert background-music here.
      Name: <NUMBER>.ogg
      (example: 47.ogg)
      File: OGG
      Background-music will loop when started till stopped.

- *game/characters*
    - Create a new folder for every new character.
      Each character folder must contain */images/* and */voice/*

    - *game/characters/NAME/images*
        - Add all images of that character here. This can be all kind of versions with different facial expressions or clothing
          Name: <NUMBER>.png *(example: 0.png)*
          Format: 640x720
          File: PNG
          You can use *game/characters/character_mask.png* as template to adjust your own images to the specification.

- *game/characters/NAME/images/big_<NUMBER>.png*
  - Add all images for the description-window in this format. The pictures should only contain the character, and no info-text.

  Name: big_<NUMBER>.png *(example: big_0.png)*
  Format: 1280x720
  File: PNG
  Please be aware that this picture will be shown, when the switch is hold in "*portrait-mode*". Anyways you have to turn the picture in 90°, because the draw function only works in a horizontal way. Please follow the example in one of the characters folders of the demo-version.
  You can use *game/characters/description_mask.png* as template to adjust your own images to the specification.

- *game/characters/NAME/images/ description_text.png*
  - This image is the overlay of the description-window. It should contain all information about your character *(example: Name, Age, Likes, Dislikes, Backstory, …).*

  Name: *description_text.png*
  Format: 1280x720
  File: PNG
  One character can only have one description_text.png
  Please be aware that this picture will be shown, when he switch is hold in "*portrait-mode*". Anyways you have to turn the picture in 90°, because the draw function only works in a horizontal way. Please follow the example in one of the characters folders of the demo-version.

    You can use *game/characters/description_mask.png* as template to adjust your own images to the specification.

- ➢ *game/characters/NAME/voice*
    - Add all voice-sound-effects of that character here.
      Name: <NUMBER>.wav
      (example: 7.wav)
      File: WAV
      You can do a full voice acting of all dialogues if you want. You can also add other sound-effects here like explosions.

- *game/lovestory*
    - ➢ This is where all internal lua files for running lovestory are. **Please do not edit any files here,** if you just want to use lovestory as it is intended.
      If you are brave and you want to change lovestory anyways feel free to do it, but I will probably give you no support if something goes wrong ☺

- *game/main.lua*
    - ➢ This is your main.lua file. It is the entry point for LovePotion. To be able to use LoveStory **you need to config a few things here**.
      Please read chapter 5.1 for more information.

- *game/novel.lua*
    - ➢ This file contains all the dialogues. If you want to write a VisualNovel you will almost **only need to edit this file**.
    - ➢ If you want to implement a dialogue-system for your story-driven game, this is the file where all conversations should go. Dialogues are saved as functions(). This means

you will be able to call them from anywhere in your game, using the static class *novel* of this file.
Please read chapter 5.2 for further information.

5. Code Documentation

In this chapter I will introduce all configurations you will need to make to run LoveStory and use it in your game. In addition I will list every function available for you. This will help you to use the full potential of LoveStory.

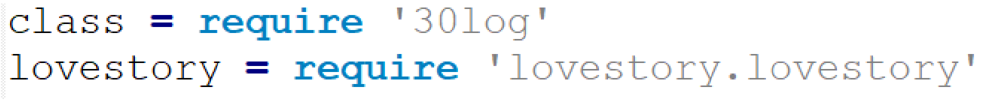
5.1 Contents of main.lua

Please add all the following things to your main.lua.

If you start a new project you can of course just use the main.lua of the demo as it is perfectly configured already. Just delete the "*preloader*" parts or use it to make your own preloader.

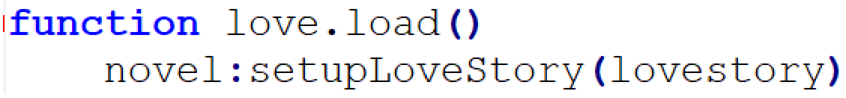
First thing you need to do is import the library using ***require.***

Because LoveStory uses the *30log.lua* library for implementing classes, your first step will be to import *30log.lua*. After that import the lovestory.lua for all lovestory functionality.



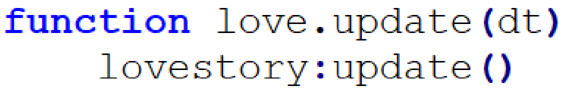
```
class = require '30log'
lovestory = require 'lovestory.lovestory'
```

In *love.load()* be sure to initialize the static *novel* class of LoveStory with *novel:setupLoveStory(lovestory)*



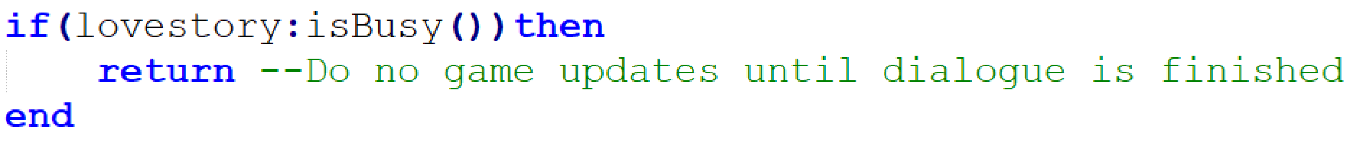
```
function love.load()
    novel:setupLoveStory(lovestory)
```

You will need to add *lovestory:update()* in your *love.update().*



```
function love.update(dt)
    lovestory:update()
```

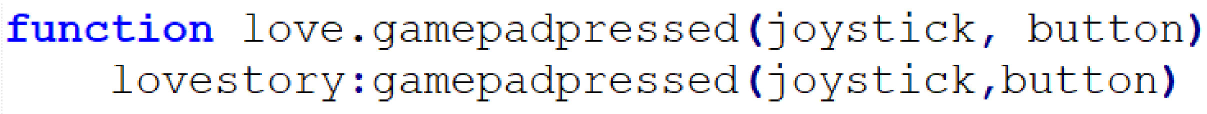
If you have other game-code to update, I recommend you check if LoveStory is running a conversation, and wait till the LoveStory dialogue is over (the player has read everything) before continuing with the game-updates. You can do it like this:



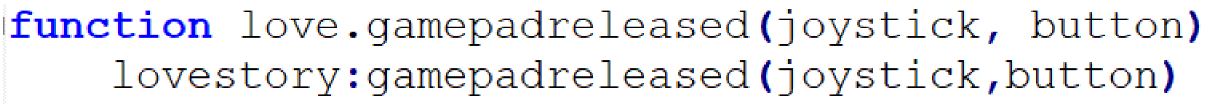
```
if(lovestory:isBusy())then
    return --Do no game updates until dialogue is finished
end
```

This way every love.update() will just be skipped till the LoveStory conversation is finished.

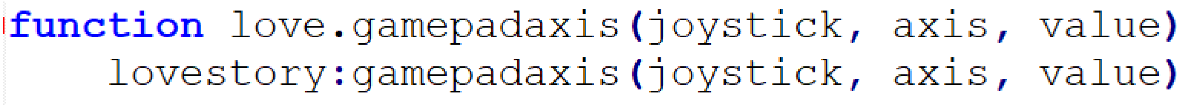
Next thing you need to do is forward all the joycon and touch inputs. You do this always the same way with every function. You will need to do this for following functions in the following way:



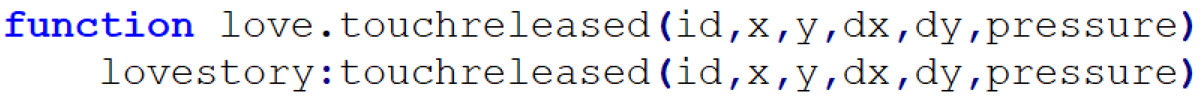
```
function love.gamepadpressed(joystick, button)
    lovestory:gamepadpressed(joystick,button)
```



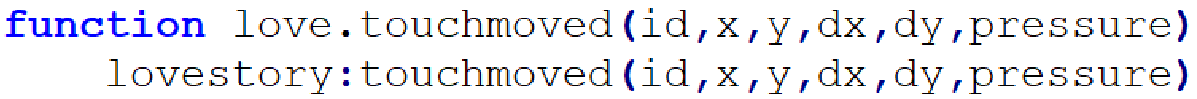
```
function love.gamepadreleased(joystick, button)
    lovestory:gamepadreleased(joystick,button)
```



```
function love.gamepadaxis(joystick, axis, value)
    lovestory:gamepadaxis(joystick, axis, value)
```

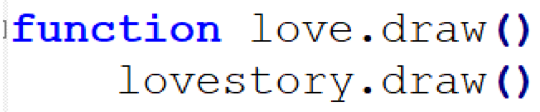


```
function love.touchreleased(id,x,y,dx,dy,pressure)
    lovestory:touchreleased(id,x,y,dx,dy,pressure)
```



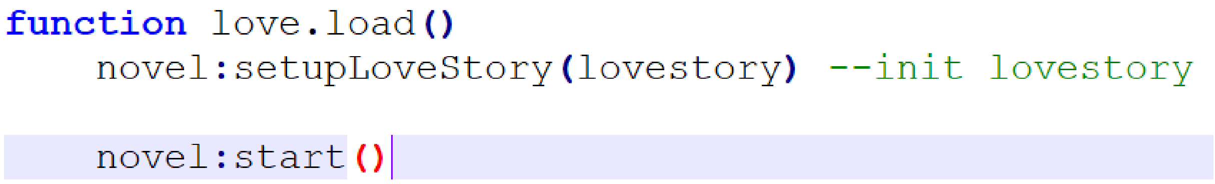
```
function love.touchmoved(id,x,y,dx,dy,pressure)
    lovestory:touchmoved(id,x,y,dx,dy,pressure)
```

The last thing you need to do is to use LoveStory’s draw() function in the main draw() function. Therefore, you just forward it like this:



```
function love.draw()
    lovestory.draw()
```

Somewhere you need to open your first dialogue or conversation. If you are building a VisualNovel, the *love.load()* would be a good place to do this. That is why in the demo it is done like this:



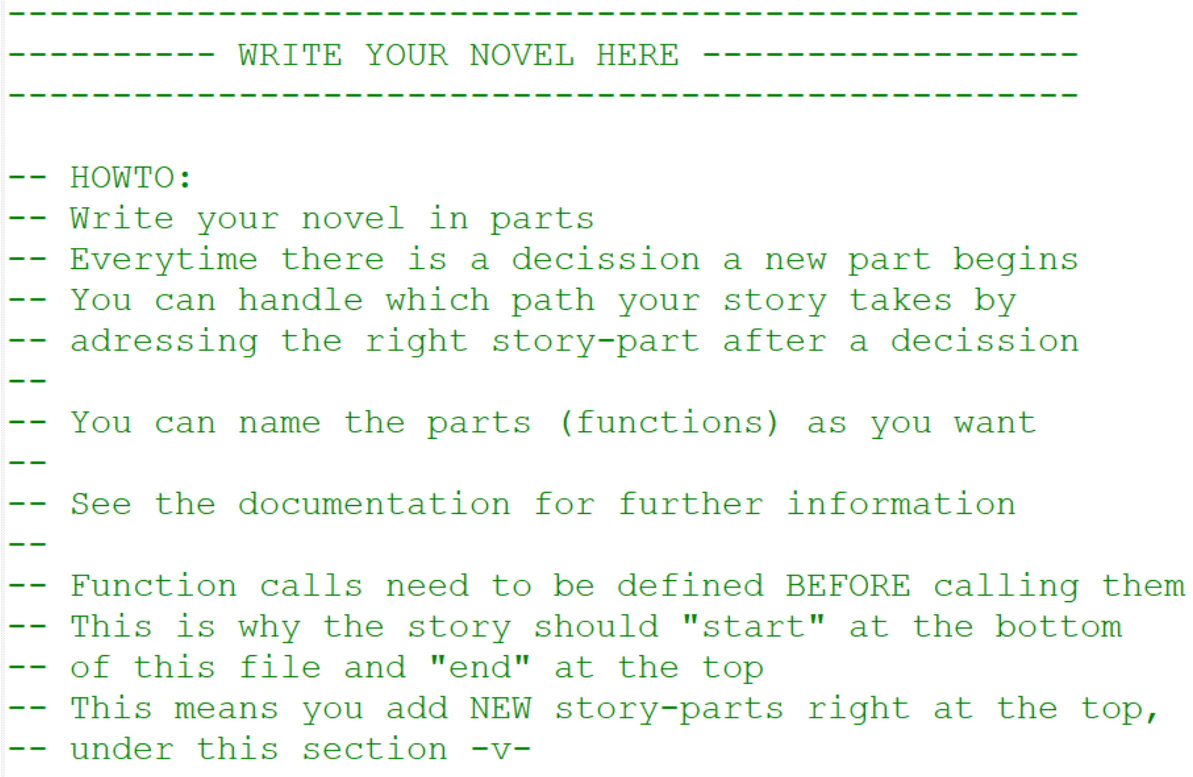
```
function love.load()
    novel:setupLoveStory(lovestory) --init lovestory

    novel:start()
```

Be aware, that *novel:start()* is no predefined LoveStory function. It is just the name of the first dialogue-function for the demo. Basically you can call it like you want. For example *novel:myFirstDialogue().* You just have to create the function it in *novel.lua.*

5.2 Contents of novel.lua

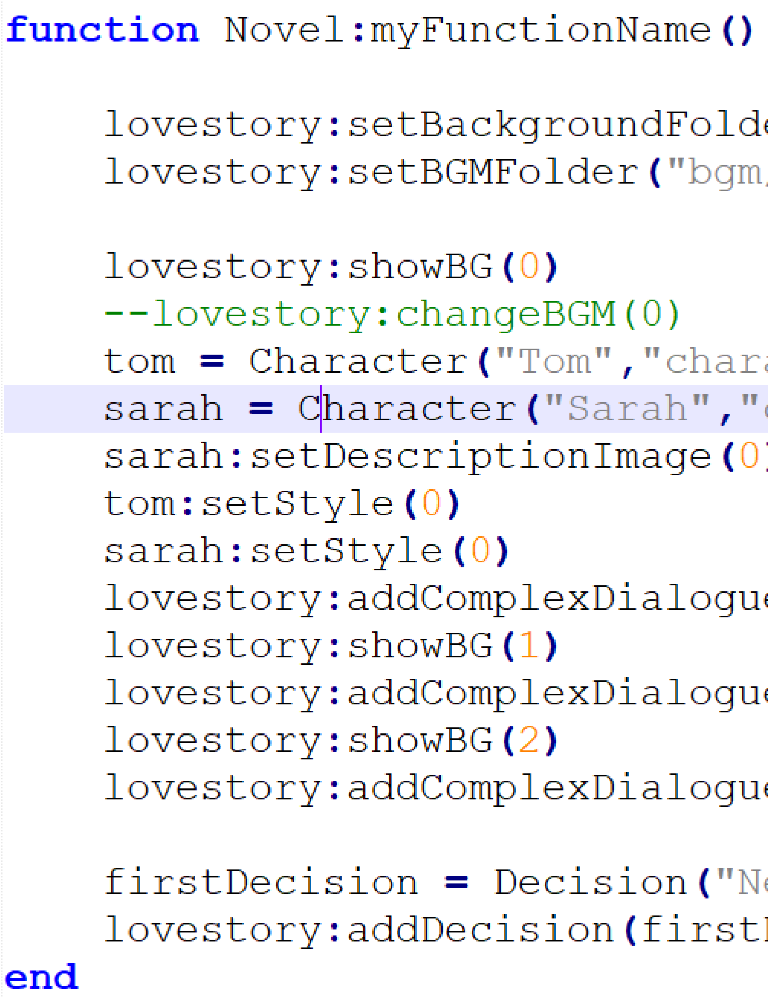
Novel.lua is the file where all your dialogues go. This is the main file you will working with when you using LoveStory to build dialogues for your game. The usage is quite easy, there are only a few things you need to know in order to build awesome dialogues with LoveStory.



```
--------------------------------------------------
---------- WRITE YOUR NOVEL HERE -----------------
--------------------------------------------------

-- HOWTO:
-- Write your novel in parts
-- Everytime there is a decission a new part begins
-- You can handle which path your story takes by
-- adressing the right story-part after a decission
--
-- You can name the parts (functions) as you want
--
-- See the documentation for further information
--
-- Function calls need to be defined BEFORE calling them
-- This is why the story should "start" at the bottom
-- of this file and "end" at the top
-- This means you add NEW story-parts right at the top,
-- under this section -v-
```

You should always create new dialogues right behind the above shown commented-lines (green). As the text above already mentions, you should write your dialogues in “parts”. These parts are programmed as functions like in the demo-example:



```
function Novel:myFunctionName()

    lovestory:setBackgroundFold
    lovestory:setBGMFolder("bgm

    lovestory:showBG(0)
    --lovestory:changeBGM(0)
    tom = Character("Tom","char
    sarah = Character("Sarah","
    sarah:setDescriptionImage(0
    tom:setStyle(0)
    sarah:setStyle(0)
    lovestory:addComplexDialogu
    lovestory:showBG(1)
    lovestory:addComplexDialogu
    lovestory:showBG(2)
    lovestory:addComplexDialogu

    firstDecision = Decision("N
    lovestory:addDecision(first
end
```

The name of the function (here “*myFunctionName*”) is your free decision and should match the plot of this part of the conversation for easy managing of your story.

How you can use every of the available lovestory:functions() will be explained in the following chapters.

Because of the way decision-making works, it is best to end the dialogue-part as soon as a decision pops up. This is because the outcome of a decision will probably start a new dialogue-part depending on what the last decision outcome was.

One more important thing: If you call a new dialogue-part from inside an existing dialogue part, you will need to define that called dialogue-part **BEFORE** the calling dialogue-part uses it (see green text).

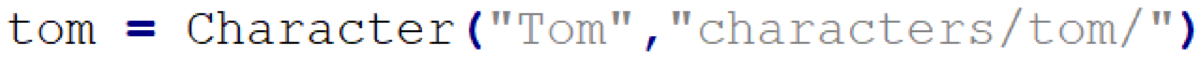
You can easy archive this, by creating new dialogues always on top of all other dialogue-parts (so right beneath the green text).

In addition, every decision needs to be defined in the *DECISION* part above the *WRITE YOUR NOVEL HERE* part. This way the decision can be addressed from every dialogue, in case you want to use the made decision answer as pre-condition for a later story path.

5.3 Characters

This chapter will explain how you can create characters and manage them. This will concentrate on the code-aspects of character creation. If you want to learn more about how to add images and voice for a character, please read chapter 4.

Characters are created as follows:



```
tom = Character("Tom","characters/tom/")
```

The first parameter is the Name of the character, which will also be shown in the top left of the text-box. The second parameter is the characters own folder, which holds his image and voice files (see chapter 4). Best is to name the character variable as the character itself, for easy managing (*tom = ….*).

Character-Functions:

`characterXYZ:setStyle(num)`

Default = -1 => No Image

This sets the characters image for emotion or clothing changes. There parameter needs to be the number used in the filename.

(images/3.png => setStyle(3))

If you want no character picture you can clear it with *setStyle(-1)*.

`characterXYZ:setDescriptionImage(num)`

Default = -1 => No Image

This sets the characters image for the special character-description screen (portrait mode).The parameter needs to be the number used in the filename.

(images/big_3.png => setDescriptionImage(3))

`characterXYZ:setName(name)`

This changes the character’s name. But only his shown name not his variable-name !!! You could use this if you want a character to be unknown for the first part of the game (like “???”).

5.4 Dialogues

This chapter will explain how to create dialogues.

Dialogues should always be defined in a dialogue-part-function in the *novel.lua* file! One dialogue always represents one text-box screen. The next dialogue begins when the player presses “A” on the joycon or touches the screen.

Dialogue Functions:

`lovestory:addDialogue(character,text)`

> The easiest way to add a dialogue.
>
> The parameters need to have a character (that defined before the dialogue) and a text, which contains what the chosen character should say.
>
> You should be sure that the text fits the dimensions of the textbox. The text should not have more than 180 characters. You have to make line breaks yourself by using “*n*” in the text-string after about 45 characters.
>
> The textbox is big enough to show 4 lines of text.

`lovestory:addEmptyDialogue()`

> This adds an empty dialogue frame.
>
> For one frame (till player presses “A” or touches the screen) there will be no characters or textbox shown on the screen. The background image will be shown!
>
> This way you can implement events where you show something in a full screen image while no textbox or character blocks the view.

`lovestory:addComplexDialogue(character,text,voice,secondCharacter)`

A more complex way to add a dialogue.

The parameters need to have a character (that defined before the dialogue) and a text, which contains what the chosen character should say.

You should be sure that the text fits the dimensions of the textbox. The text should not have more than 180 characters. You have to make line breaks yourself by using “*n*” in the text-string after about 45 characters.

The textbox is big enough to show 4 lines of text.

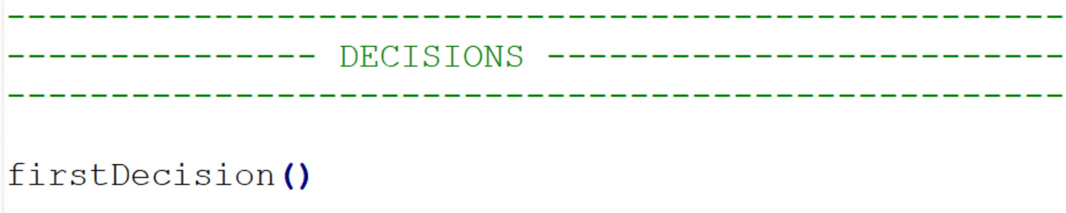
In addition to that, you can add a voice file. The *voice* parameter needs to be a number corresponding to the wav file in the /voice/ folder.

With the *secondCharacter* parameter you can add a second character, which will be drawn on the left side of the screen.

5.5 Decisions

This chapter will explain how to create decisions.

Decisions must be created under the *DECISION* comment in *novel.lua*, but it needs to be defined when you first show it.



```
-----------------------------------------------------
---------------- DECISIONS --------------------------
-----------------------------------------------------

firstDecision()
```

When defining a decision you can add from 1 to 4 options for the player to choose (yeah only one works, too ☺).

There is no way to cancel a decision, except you program one of the options for cancelling!

This options can be combined with so called "*callback functions*". This functions will run when they are chosen. In theory you can add every function you want (so maybe for advanced game-designers this is something cool), but for now let's just assume you use only other dialogue-part-functions from *novel.lua* here.

This way you can control the flow your story takes. When the player takes *option1* he will get *novel:apple()* if he takes option2 he will get *novel:grape()* and so on. Take a look at the example:

```
firstDecision:define("Yeah",self.apple,"No",self.banana,
                              ,"I dont care!",self.banana,"",nil)

lovestory:addDecision(firstDecision)
```

Be careful to type **self.***myDialogueClass* with only a point and <u>no colon</u> (!!!) and without the brackets "*()*" at the end!

This is because you give the function as a variable!

Of course, if you use "*self*" the target function must be placed in *novel.lua.*

After the decision is added, you should **end the current dialogue-part**, because the callback functions will make the story continue in other dialogue-parts and so it **makes no sense** to add further dialogue elements.

Decision Functions:

`lovestory:addDecision(decision)`

> Adds a decision to the dialogue. You should **end the current dialogue-part**, because the callback functions will make the story continue in other dialogue-parts and so it makes no sense to add further dialogue elements.

`optionCount = decisionXYZ:getOptionCount()`

> This function returns the number of options available for the given decision.

`optionText = decisionXYZ:getOption(i)`

> This function returns the text of option i for the given decision.

`answerNumber = decisionXYZ:getAnswer()`

> This function returns the number of the option the player has chosen (1 to 4). If there was no answer given until now, the function will return the number 0.

5.6 Background Images and Music

This chapter will explain how to create and change background images and music.

By default the folders for **music** and background **images** will be */backgrounds/* for images and */bgm/* for background-music, but you can change them if you want with the corresponding functions.

You can read about the format and specifications of the files in chapter 4.

Background Images and Music Functions:

`lovestory:bgmChange(num)`

> `Default = -1 => No Music`
>
> This sets the current background music. The parameter needs to be the number used in the filename.
>
> `(bgm/3.ogg => bgmChange(3))`
>
> The music will be **looped** until it is stopped, or the music is changed again.
>
> You can **stop** the music with *bgmChange(-1)*.

`lovestory:showBG(num)`

> `Default = -1 => No Image`
>
> This sets the current background image. The parameter needs to be the number used in the filename.
>
> `(backgrounds/3.png => bgmChange(3))`
>
> If you want no background you can clear it with *showBG(-1)*.

`lovestory:setBackgroundFolder(path)`

Default = /backgrounds/

You can change the default background-image folder to any path you want.

`lovestory:setBGMFolder(path)`

Default = /bgm/

You can change the default background-music folder to any path you want.

6. Known Bugs and Future Improvements

Because LovePotion is still in development there is not yet every function available I would need to build LoveStory as I would like to.

But I will continue to implement this functionalities as soon as they are available.

Never the less I would like to tell you what is not possible at the moment ☹ …

- **The Portrait mode and special character-description screen:**

  I planned, that the special character-description screen will pop up every time you hold your switch in portable-mode in a 90° angle. This way it would be ensured that you only could see the picture shown the “right way”. Sadly, we cannot read the sensors of the joycon-gyroscope now (at least with LovePotion).

  In addition to that I wanted to make the text shown in the character-description screen to be editable through code, but we can not print text in a 90° angle at the moment. That is why I decided to add the text as an additional png image instead of text-objects.

  However, as soon as these functionalities will come I will implement them. Until then you will need to press the R-Key (right shoulder button) to activate this screen and use the “*description_text.png”* to edit the shown text. ☹

- **The second character on left screen side**

  At one moment, I thought about implementing the second character on the left side of the screen with an image-flip on the vertical line, but LovePotion cannot do that at the moment because of lacking GPU support of the switch.

I would have done that, because the character is then more aligned to the side of the screen without using an extra image or shifting the image an unknown offset-size to the left.

But after overthinking it, maybe a mirrored character picture would irritate people even more… (?)

7. Thanks and Acknowledgements

This chapter is for all the people that motivate and help me in writing things for Nintendo Switch and LovePotion. Thank you all for using my stuff and reading through this whole documentation. I want to mention some people here that helped me a lot, everyone in their own way ☺

- The whole "*Tiny Turtle Industries*" Discord Server:

  Thank you so much guys! You are just awesome! A great community, and not only the LovePotion talk is great with you ☺

- TurtleP:
  Dude you are so frkn awesome! You are porting Love2D to the Switch, what is the basic of all this work! Moreover, you are "technical support guy" not only for me, but also for most of the other LovePotion developers, too. I could always ask you when I needed help, and you always helped me.☺

- Mik:
  Thanks again for the great artwork you made for my Snake-Port (Version 4). Now the game finally does not look like garbage anymore. Your design and art skills are badass! Keep up the good work! ☺

- My coworker Max:
  Talking with you about the switch hacking scene is always big fun. When I show you stuff I programmed you are always interested and give me new ideas. Work would be much more boring without you dude! Thanks for that! ☺