I had to change a few more things to deal with problems/desires Jordan Cramer mentioned. They should not affect inventory or save/load except that we may need to create three inventory objects.

First off a command table (Text objects need to be able to do these things. We need to be able to remove things from the inventory now.):

|  |  |
| --- | --- |
| Command | Meaning |
| next | End the menu and goto next event  (Event will handle this) |
| exit | Exit the game  (The mainloop handles this) |
| new | Restart the game  (mainloop) |
| omit  *Don’t rely extensively on this.* | Omit this menu item next round if *again* is used  (text, alone) |
| forget  *Don’t rely extensively on this.* | Forget this menu item permanently  inventory.add(“menu:menuname:typedchoice”)  ->The integer is the same as the index of the choice in the menu. The stored menuname is generated by Event. |
| again | Repeat the menu without leaving the Text |
| +item | Add item to inventory |
| -item | Remove item from inventory |
| \_event | Goto event and end this event.  **Events contain underscores. Underscore position is an example.**  (event handles this string) |
| ~event  *Don’t rely extensively on this.* | Goto event while leaving the text after the menu for after the event. This means that the event class will eventually contain a modified copy of the mainloop.  (event handles this) |
| “This will appear on the screen if this choice is made.” | The double quote allows text to be shown. |

General file structure:

Rules:

1. Let there be start tags <tag> and end tags </tag>
2. Tags **never** get sent to Text objects
3. Tags are conditions and conditions only, with three exceptions:
4. <!>a comment isn’t sent to Text objects</!>
5. <nl>Here newline characters are sent to text objects</nl>
6. This tag adds a newline character: <el>
7. This tag adds a space character: <sp>
8. Extra end tags are ignored.

**Events must have an underscore \_** (Not a dash) since this is how they will be identified as not being in the inventory. Inventory items should only contain letters and numbers. This are used as the conditions within the tags. A condition can have an exclamation mark ! at the beginning to invert it.

Examples:

Events Only:

<\_boardroom>Visited the boardroom.<!\_cupboard>Visited the boardroom and the cupboard</!\_cupboard><!\_boardroom>This will never show up</!\_boardroom></\_boardroom>

Items Only:

<chainsaw><!apple>

<nl>Real men start their chainsaws with an apple.

The chainsaw make a sputtering sound and dies a slow death.</nl>

</!apple><apple>

<nl>You yank that chain and hear the chainsaw go to a roar.<nl></chainsaw>

You then eat the sliced apple.</apple>

Special ways that text objects will handle text sent to them:

There are three kinds of text they can get: An inline command, a menu, or plaintext. They will never get tags and will only ever need to check menu conditions.

**The major change to text objects is that they will need to store a vector of conditions.**

Plaintext:

Anything that is not the other two.

Inline Commands:

~+item ~-item

-> Check for the tilda then process the command (see the table at the beginning)

-> Anything that makes sense from the table at the beginning

Menus

Menus begin with a question mark. Once they reach a text, the syntax is like this:

?menunamehere^The question contains every word,

and every line

that the user must see

to choose an answer^

#choose#answer^command command

Anothercommand

NoticeTheWhitespace

#42#I know the meaning of life^ +fourtytwo \_heaven +kitty

Note: the second option would add fourtytwo to the inventory *and then* goto heaven.

The text object will have to generate its own list of what the user can type.

Summary:

? at the beginning means it is a menu

^ ends the question or the answer, depending on the context. It also splits the menuname and question.

# Starts the answer and breaks up the ***typed choice***. The answer is followed by the commands associated with it.

The commands need to be done left to right even if it means that the kitty will never go to heaven. Commands are separated by spaces. See istringstream for treating a string as input. ***The omit and forget commands added some complexity to the text object*.**

In terms of **writing** it:

Tags can be inserted *anywhere*.

BEGIN

It is possible to make writing menus less ugly since they *never* get sent to Texts:

?</nl>

<!>The opposite of the always-true condition lets me comment</!>

<!>There are multiple lines, but they never show up by the time this makes it to Text.</!>

<!>Make sure to leave whitespace if newlines are turned off</!>

Why did the chicken cross the road?^<!apple><el></!apple>

#1#Stare blankly <apple>and eat your apple</apple>^

<apple>-apple</apple><sp>

“GAME<el>OVER”<sp>

exit<sp>

#2#Think Longer^ again

END

When the menu reaches a text, it would could look like this:

START

?Why did the chicken cross the road?^#1#Stare blankly and eat your apple^-apple “GAME

OVER” exit #2#Think Longer^ again

END

Or this:

START

?Why did the chicken cross the road?^

#1#Stare blankly^”GAME

OVER” exit #2#Think Longer^ again

END

Because the possible user entries have to be typed manually, there is no worry about a tag eliminating an answer. ***This can be used as an alternative to omit and forget if Bhupender thinks it will take too long to implement them in Text. Adding an inventory item and reloading would achieve the behaviour Jordan Cramer talked about with less code.***