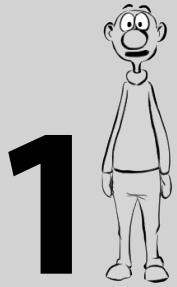
Tutorial:
Building a Save/Load Menu in Visionaire Studio 5

This tutorial explains the creation of a save/load menu on the basis of a little demo game. The game project zip file comes with 4 different .ved files that build on each other – from the absolute basics to a menu with some bells & whistles:

It's assumed that you have worked with the Visionaire editor before.



Bare Necessities

Disclaimer: It is often possible to do things in Visionaire Studio in multiple ways. This is no extensive manual that covers all possibilities. It is just a guide to show how I do things.

– Esmeralda

1. Bare Necessities

There is not much needed to get a working save/load menu, if you strip away feedback for the player and amenities: **save slots** and **buttons** to trigger saving and loading.

1.1 The menu scene

First let's add a menu. Since creating save slots isn't possible on regular scenes, it has to be a "Menu" type scene.

► In "Scenes", click on the green [+] and choose "Menu" (Fig. 1-1).

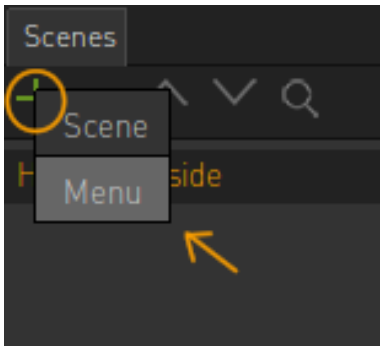


Fig. 1-1: Add a menu

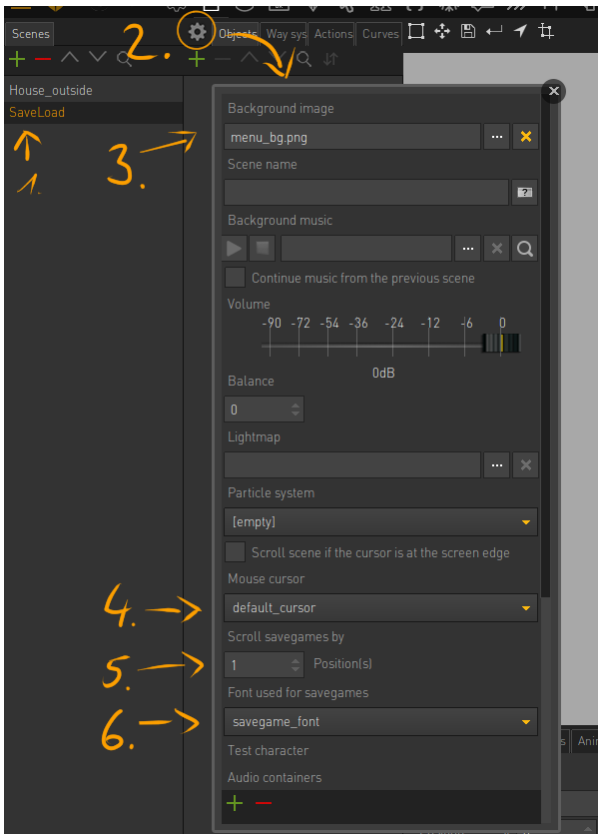


Fig. 1-2: Set up the menu properties

Next step is setting up the menu properties (Fig. 1-2):

► 1. Name your menu (I called it "SaveLoad").

► 2. Open the scene properties (gear icon).

► 3. Add the background image of the menu. In the demo project it's called "menu_bg.png" (located in the "Menu" folder).

► 4. Choose the menu cursor.

► 5. "Scroll savegames by" defines by how many slots the savegames will move when you click on a left/right (or up/down) arrow.
We will make use of this feature in our "Basic Version" (chapter 2). Keep the default value (1) for the demo game.

► 6. Select a savegame font.
If you select a font here, the engine will automatically display a timestamp underneath the save slot. If you don't want that, just leave the field empty. I created a font called "savegame_font" that's a little smaller than the regular game font.

1.2 Save slots

Save slots (aka. savegame areas) are rectangular areas on the menu to hold the savegames. Each slot will show an in-game screenshot when used/occupied.

Now how do the save slots exactly work in the game?

To save a game the player has to select a save slot first by clicking on it. The engine will register that and store the internal slot number to execute a save (or load, delete) action on it later.

Note that the player won't get a visible reaction when hovering over or clicking on a save slot, neither from the cursor nor from the slot itself.

After having chosen the save slot, the player has to click on a separate "save" button to execute the save action. The engine will save the game, take a screenshot of the scene the current character is in and place that as a thumbnail in the chosen slot.

However, if there are other free slots prior to the chosen one, the engine will automatically move the savegame to the first free slot to leave no gaps.

If the chosen slot is already occupied, the existing savegame will be overwritten.

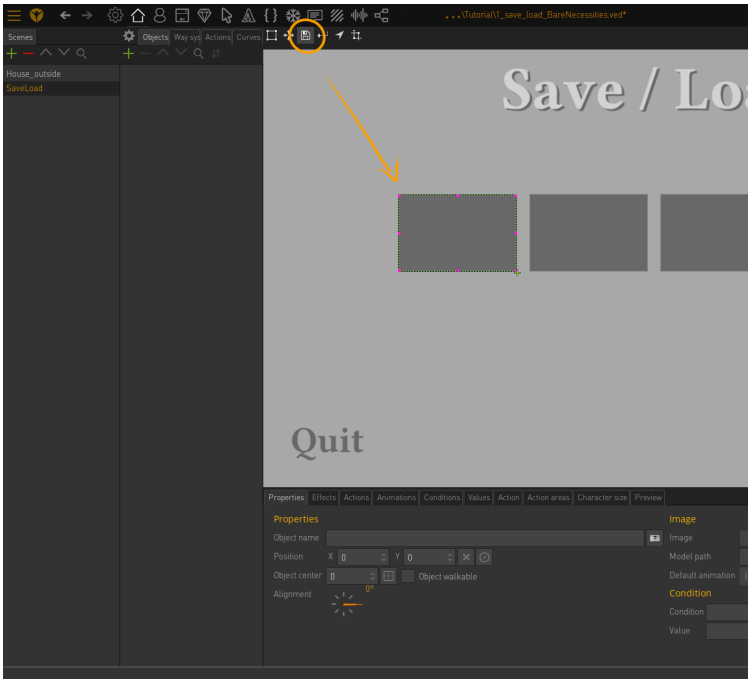


Fig. 1-3: Draw the save slots

Let's draw the save slots (Fig. 1-3):

- Click on the floppy disk icon.
- Draw the first save slot rectangle onto the menu by clicking and dragging while holding down the mouse button.

Note that the size of your first save slot will define the thumbnail size for all other slots. So it is not possible to have different sized thumbnails, even if you draw the following slots smaller or bigger.

The save slots are numbered in the order they are created. That means that you determine the order the save slots get filled by the order you draw them.

To avoid confusion for the player I recommend sticking to the traditional order (as I did in the demo game, Fig. 1-4). 🙂

- With that in mind, go on and draw the remaining save slots.

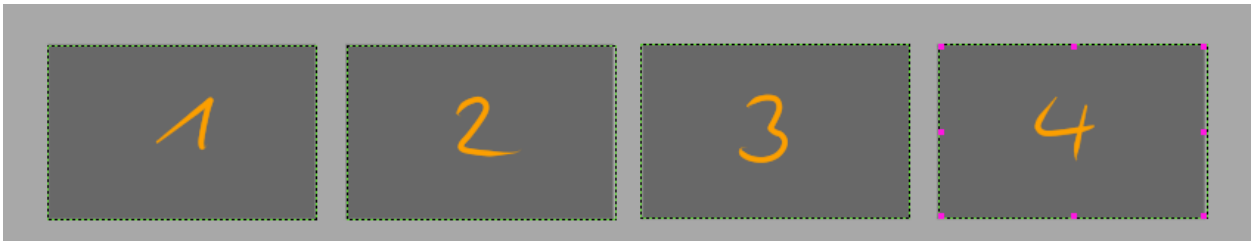


Fig. 1-4: Save slots 1, 2, 3, and 4 – in a somewhat boring yet distinct order

1.3 Save, Load and Delete buttons

Now we will create the buttons that execute the actions on the save slots. We'll start with "save" (Fig. 1-5).

- 1. Add a new entry to the menu's "Objects" list and name it (e. g. "btn_save").
- 2. Add the button image.
- 3. Click on the 4-arrows icon and drag the button image to its position.

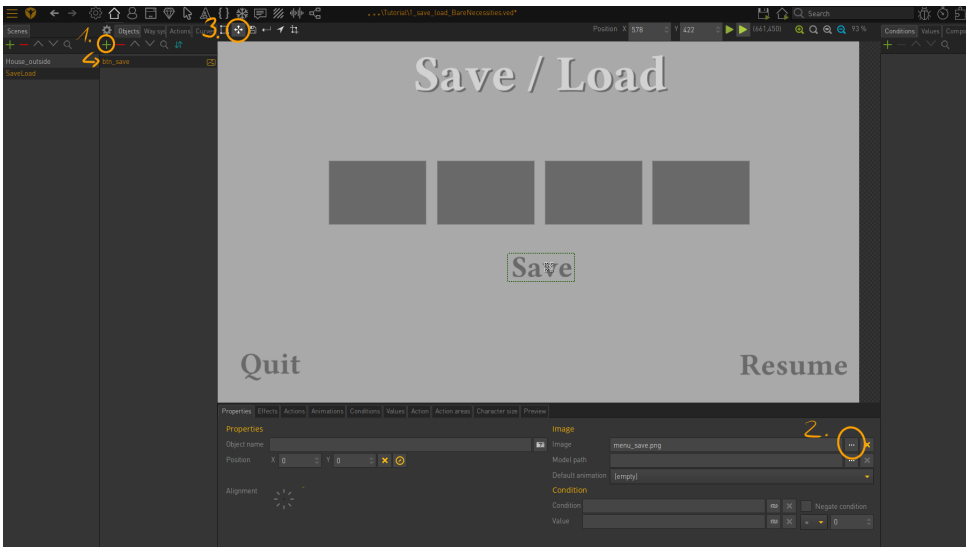


Fig. 1-5: Create the save button

Fig. 1-6:

- 4. Click on the area icon and draw the object area around the button.
- 5. Add an action for the button in the "Actions" tab and select "Left click" as its execution type.
- 6. Add the action part "Load /Save game" from the "Savegame" category.
- 7. In the action part, select the "Save" option.

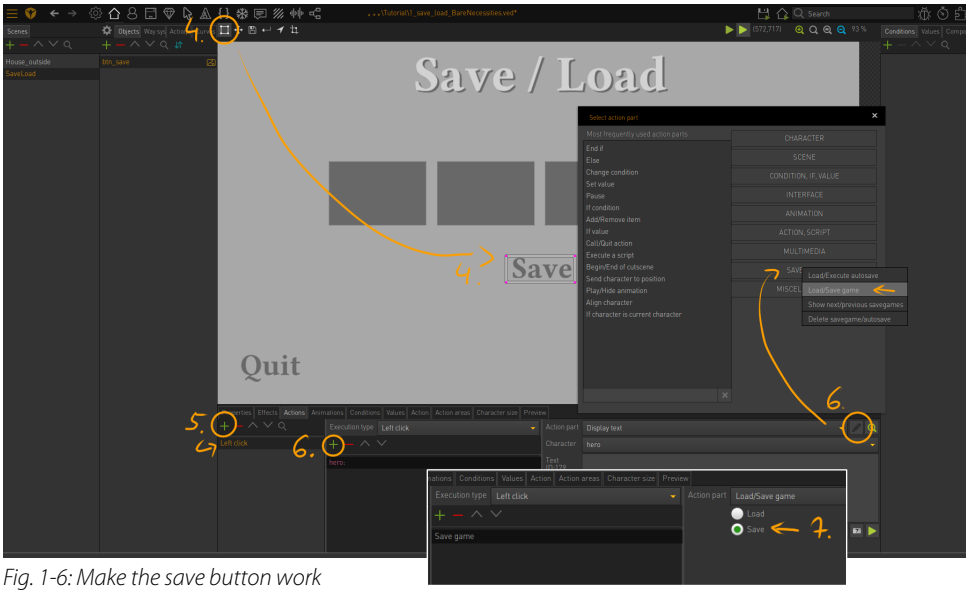


Fig. 1-6: Make the save button work

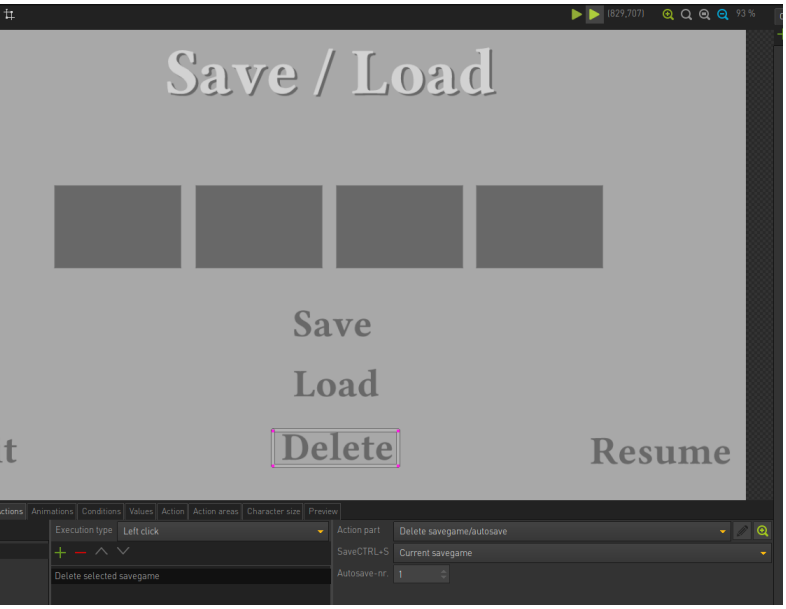


Fig. 1-7: Add the delete button

- Now repeat these steps for the "Load" and "Delete" buttons.

In short: Create new entries in the "Objects" list, name them, add the images and place them. Draw the objects areas and add a "Left click" action.

To save some time, you could also duplicate the save button and adjust the settings (don't forget to move the object area, too, with Ctrl+drag though).

- In the "Left click" action for "Load", add the action part "Load/Save game" again, but select the "Load" option.
- For the "Delete" button add the action part "Delete savegame/autosave" instead and select "Current savegame" in the dropdown (Fig. 1-7).

That's basically all that's really necessary to save, load and delete a savegame.

1.4 More menu functions

► To finish our menu, let's add buttons to quit and resume the game.

You know the drill, create new entries in the "Objects" list, name them and draw the object areas. I painted the "Quit" and "Resume" buttons onto the menu background, so no need to add object images in this example.

► In the "Quit" button action add the action part "Quit game" from the "Miscellaneous" category (Fig 1-8).

To continue playing our game after saving, we need to have a possibility to go back to the scene of our main character.

► Add the action part "Change to scene of a character" to our "Resume" button and select "Current character" in the dropdown (Fig. 1-9).

You could choose your main character by name, if you only have one playable character, but "current character" will work in any case.

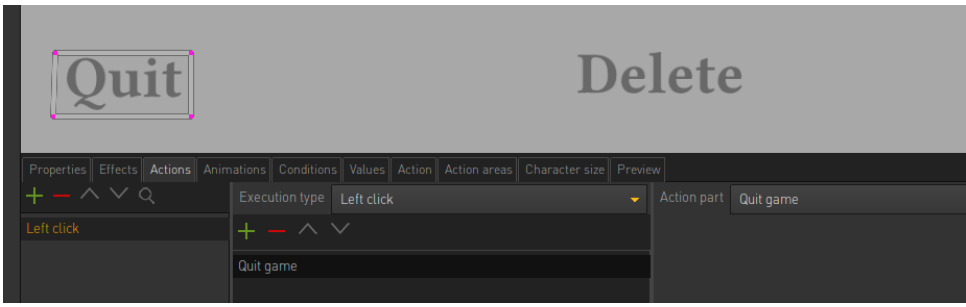


Fig. 1-8: Add the quit button

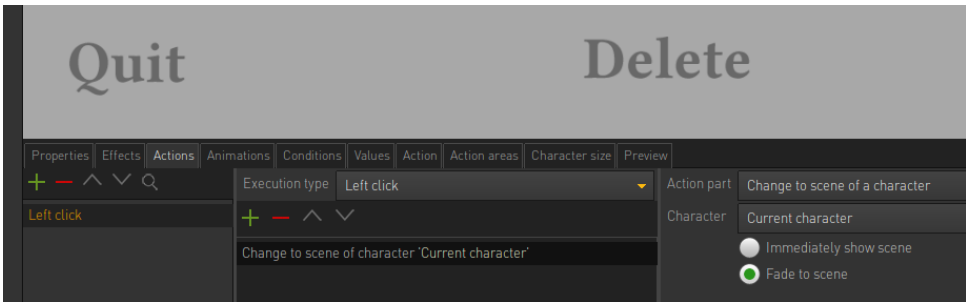


Fig. 1-9: Add the resume button

1.5 Open the menu

To open the save/load menu in-game we need a button or key action. I like adding a "Menu" button to my inventory. But of course you can do it however you like: a button on a different interface, an action in the "Key actions" (Game properties) for the "Esc" key …

► In the action of your choice, add an action part "Show scene/menu" from the "Scene" category (Fig. 1-10).

► In the action part select our save/load menu from the second dropdown.

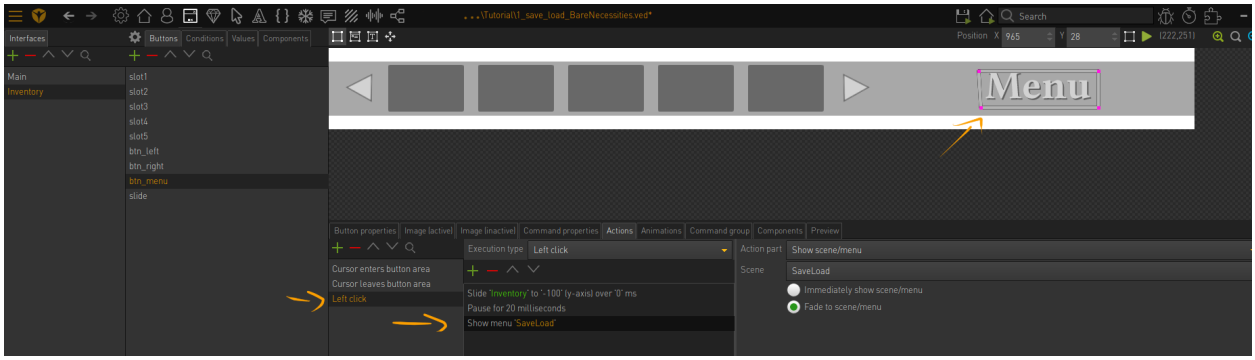


Fig. 1-10: Action to show the save/load menu

In the demo game I use an inventory that slides in from the top (through a special plugin). To click on the menu button it needs to be fully visible. But I don't want it to appear on the savegame thumbnail and I don't want it to be visible when the savegame is loaded. So I need to slide it out prior to showing the menu. That's why I added an action to slide it up again and a small pause to give the engine time to execute before changing to the mene scene.

1.6 Testing

Let's do a little test.

- Run the game and click on the button to call the menu scene (in case of the demo game that's inside the inventory).

Our menu will show up (Fig. 1-11).

- 1. Click on a save slot (this gets registered by the engine, even if we can't see any reaction).
- 2. Click on the "Save" button.

Fig. 1-12: It worked. 😍

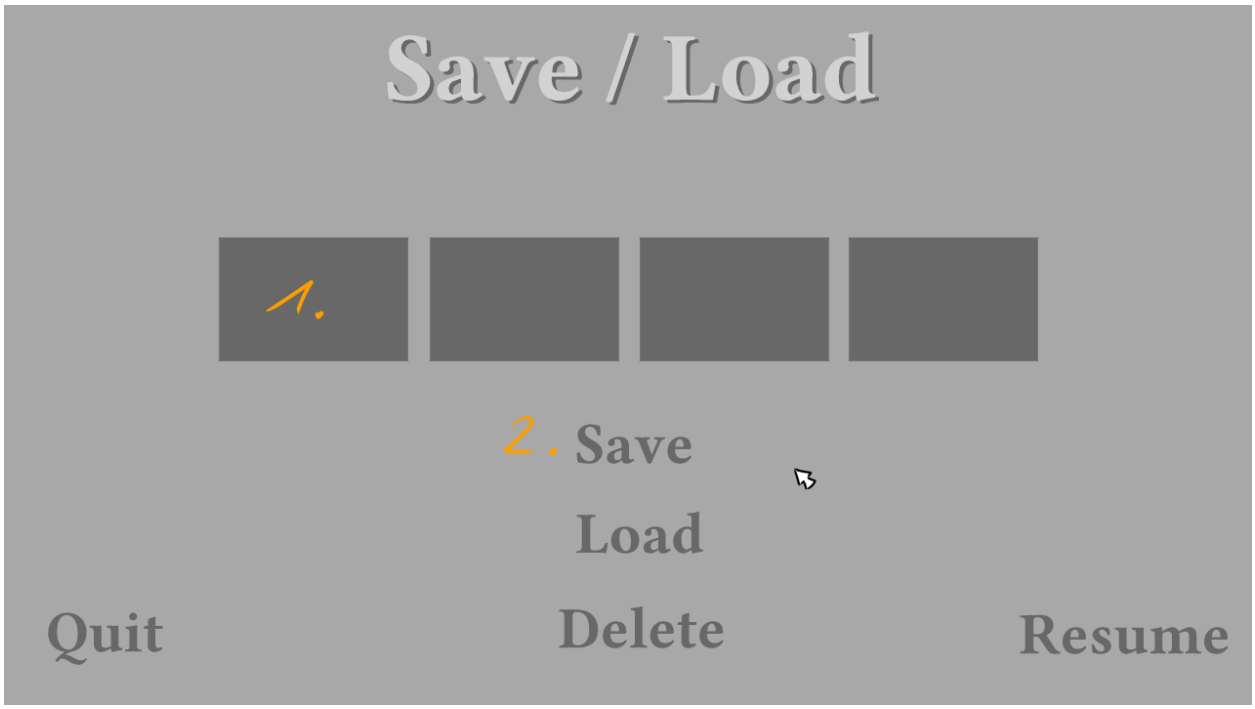


Fig. 1-11: Save the game

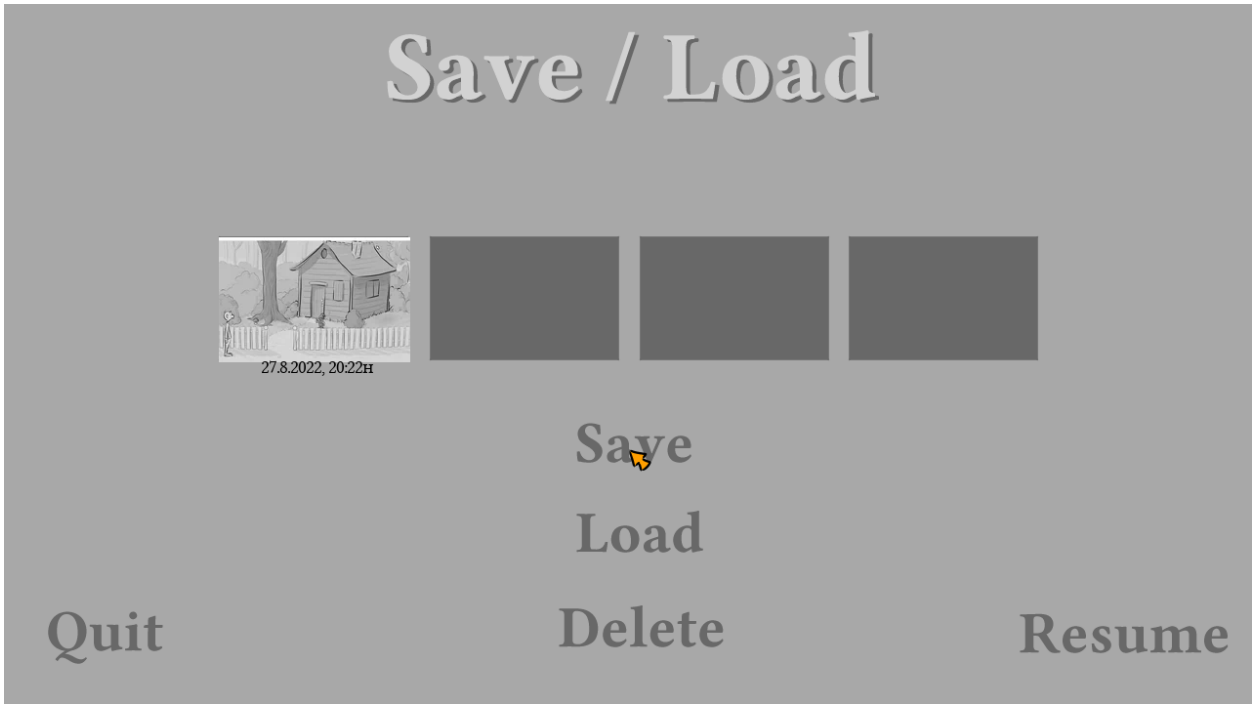


Fig. 1-12: Upon saving, a screenshot thumbnail apperas in the save slot along with a timestamp

Basic Version

2. Basic Version

Now we got a working save/load menu, but it is not very intuitive to use yet. How should the player know that they need to click on a slot first, how should they know if a savegame is selected? And besides, we only have 4 save slots …

In this "Basic Version" we develop the menu further and

- add highlighting to the save slots,
- make the buttons responsive,
- add arrows to scroll through the (then unlimited number of) savegames.

2.1 Infinite save slots

Let's start with making the amount of save slots infinite by adding arrows to scroll through the savegames.

Left button

- 1. Add a new object, call it "btn_left", add an image to it and place it (Fig 2-1).
- 2. Draw an object area.
- 3. Create a "Left click" action with the action part "Show next/previous savegames" and select "Previous".

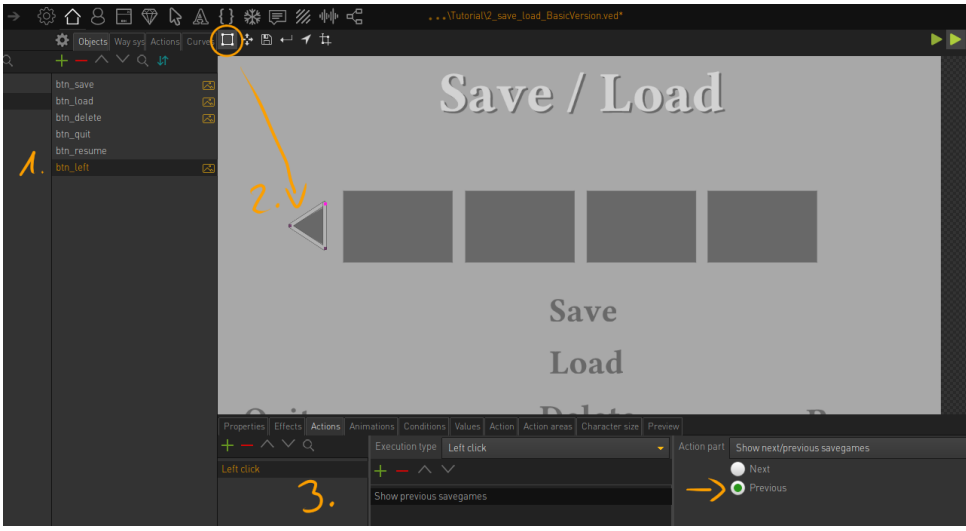


Fig. 2-1: Add a savegame scroll button

We want the scroll arrow to highlight when hovering over it. There are multiple ways to show different states of a button.

You can use two objects, each holding a different image of the button and toggle between these objects by linking them to a condition or a value. Or you can change the opacity of an image with the action part "Set object visibility". You can even force the engine to play a certain frame of an animation linked to that object to display a specific image (that would require some lines of code though).

I will mainly use values and object visibility in this tutorial.

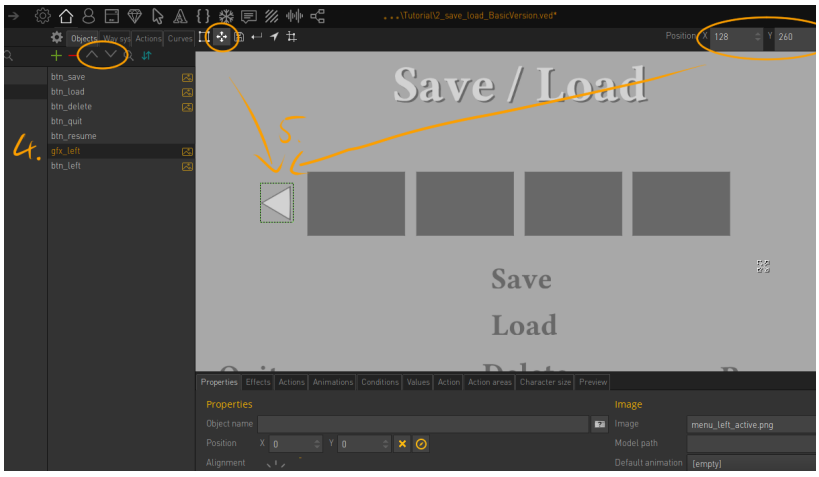


Fig. 2-2: Add a second object for the highlight (active) state of the button

Fig. 2-2:

- 4. Create a second object called "gfx_left" and add the active button image ("inventory_left_active.png"). *This object will not be clickable, that's why I use the "gfx_" prefix. This naming convention is of course optional.*
- 5. Place the image in the same spot as the inactive one. Use the coordinates at the top to do it precisely.

The active image needs to be drawn above the inactive image to cover it. There are no object centers in "Menu" type scenes, so the order in which the images are drawn solely depends on the order in the objects list.

- Move the second object above the first one (if necessary) by using the up/down arrow icons.

Currently only the active state of the button would show, because it's covering the other one. We need to hide the active image by default and show it only when hovering over the button (Fig. 2-3).

- Add two actions to the "btn_left" object – one with the execution type "Cursor enters object area", one with "Cursor leaves object area".
- For "Cursor enters object area", add the action part "Set object visibility", select the "gfx_left" object and set the visibility to 100 %.
- Do the same for "Cursor leaves object area", but set the visibility to 0 %.

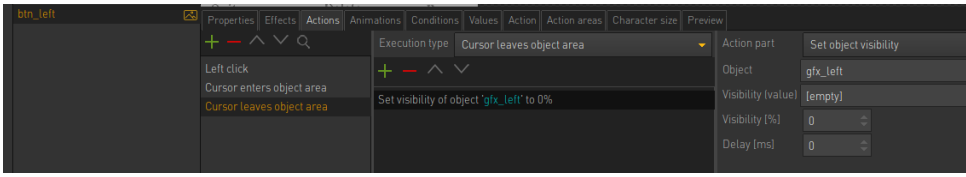


Fig. 2-3: The active button visibility depends on the cursor being moved over the object

Right button

Now that the left scrolling button is working, we have to do it all again for the right button.

- Create the right scrolling arrow just like the left one: select the right arrow images, choose "Next" instead of "Previous" in step 3, and select the "gfx_right" object when setting the object visibility.

The arrows will scroll the savegames by 1 to the left/right, because we didn't change the default value for the "Scroll savegames by" parameter in the menu properties (see chapter 1.1, step 5). In case you have other set-ups of your savegame menu and want to scroll by an entire row or column, you can set the corresponding number in the properties.

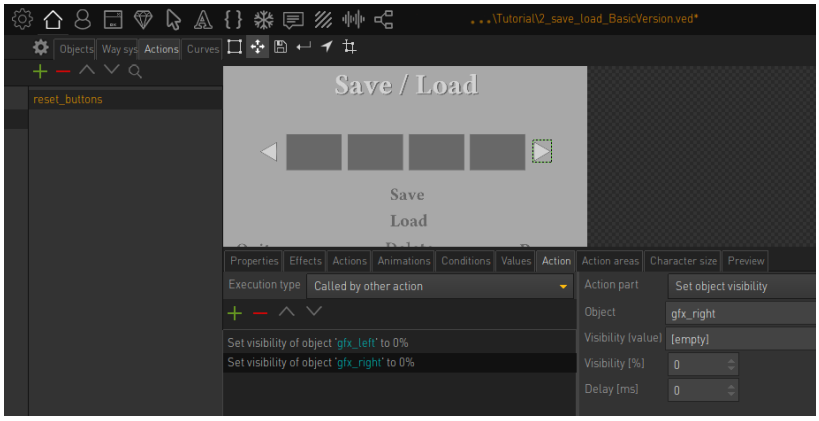


Fig. 2-4: Reset the button states

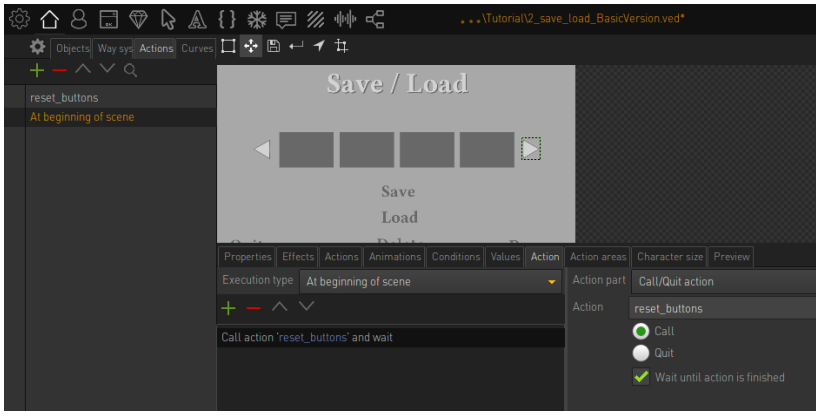


Fig. 2-5: Call the reset action when entering the menu

Reset the buttons

As already mentioned, the active state images need to be hidden by default. We have to set their visibility to 0 % at the beginning of the scene, else they would be active when calling the menu for the first time.

- Create a "Called by other action" action in the scene's "Actions" tab (Fig 2-4).
- Name the action "reset_buttons".
- Add two action parts, setting the visibility of "gfx_left" and "gfx_right" to 0 %.

We will add a few more of these action parts to that action to reset other buttons later.

- Add an action with the execution type "At beginning of scene" (Fig 2-5).
- Add the action part "Call/Quit action" and link it to the "reset_button" action with the "Call" option selected.

2.2 Enhance other buttons

Now we want to add some features to buttons we already created in the "Bare Necessities" version.

Hide buttons by condition

The save, load and delete buttons should only show up when they can be used.

- Add two conditions to the menu and call them "save_possible?" and "load_delete_possible?", both with an initial state of "False" (Fig 2-6).
- Select the "btn_save" object and link it to the "save_possible?" condition in the object's properties.

The button will now only be visible and clickable when the "save_possible?" condition is true. Do the same for the load and delete buttons.

- Link both the "btn_load" and the "btn_delete" buttons to the "load_delete_possible?" condition. They will be visible and clickable when the condition "load_delete_possible?" is true.

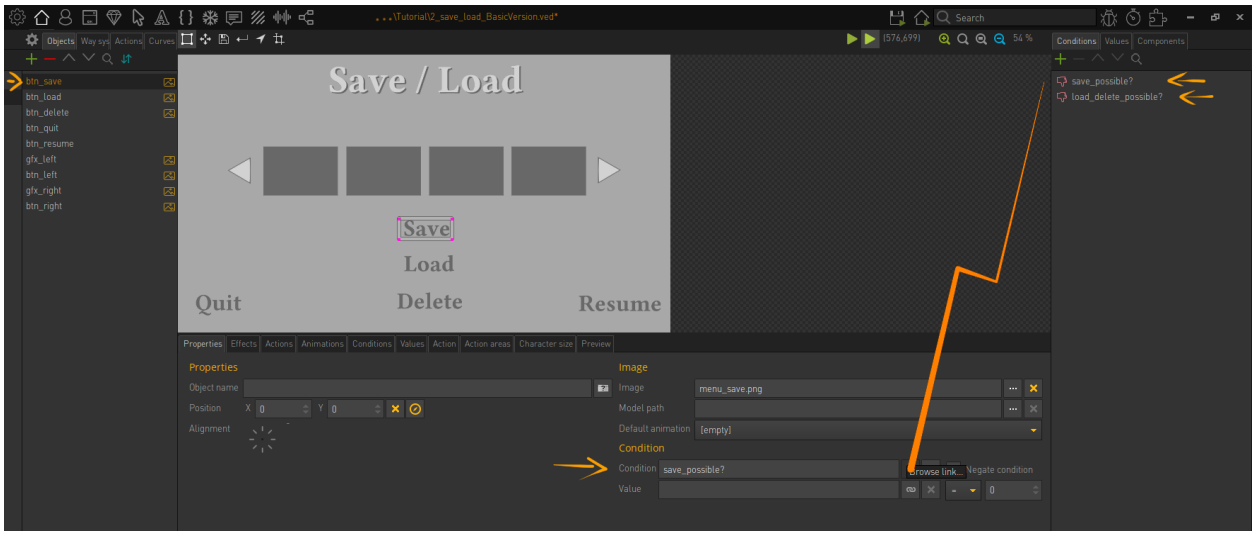


Fig. 2-6: Linking an object to a condition

Highlight the buttons

To give visible feedback when hovering over the buttons, let's do the same as for the arrows:

- Add objects for the three buttons called "gfx_save", "gfx_load" and "gfx_delete" that hold the active image of these buttons.
- Take care these "gfx_" objects are above the respective "btn_" objects in the objects list.
- Add "Cursor enters object area" and "Cursor leaves object area" actions to the "btn_" objects, setting the corresponding "gfx_" object's visibilities to 100 % and 0 %, respectively.
- In the "Left click" action of the save and delete buttons add the action part "Call/Quit action" and link the "reset_button" action (Fig. 2-7).

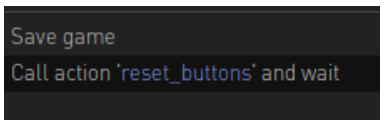


Fig. 2-7: Call the reset action after saving

Unlike for the save and delete buttons, there is no need to reset the buttons after clicking on "Load", because we instantly leave the menu upon loading a savegame.

- In the "reset_buttons" action add action parts to set the new objects "gfx_save", "gfx_load" and "gfx_delete" to 0 % visibility (Fig. 2-8).
- Also in the "reset_buttons" action change the conditions "save_possible?" and "load_delete_possible?" back to "False".

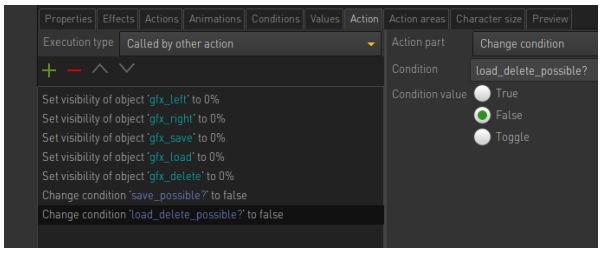


Fig. 2-8: Reset all button visibilities and conditions

2.3 Highlight the save slots

A save slot should show a visible reaction when clicked on, so the player knows this is the chosen slot. Since the slots themselves don't have the ability to do anything besides holding a savegame, we have to add objects on top of them, which we can interact with.

Also, we want our save, load, and delete buttons to be visible only if it makes sense to have them available. Until now, the "save_possible?" and "load_delete_possible?" conditions are not doing much, because they stay false and thus our three buttons stay hidden. If the player selects an already used save slot, they should have the option to load or delete that savegame, whereas we don't want to show these two buttons when an empty slot is chosen. The save button should show up whenever a slot is selected (if occupied or not).

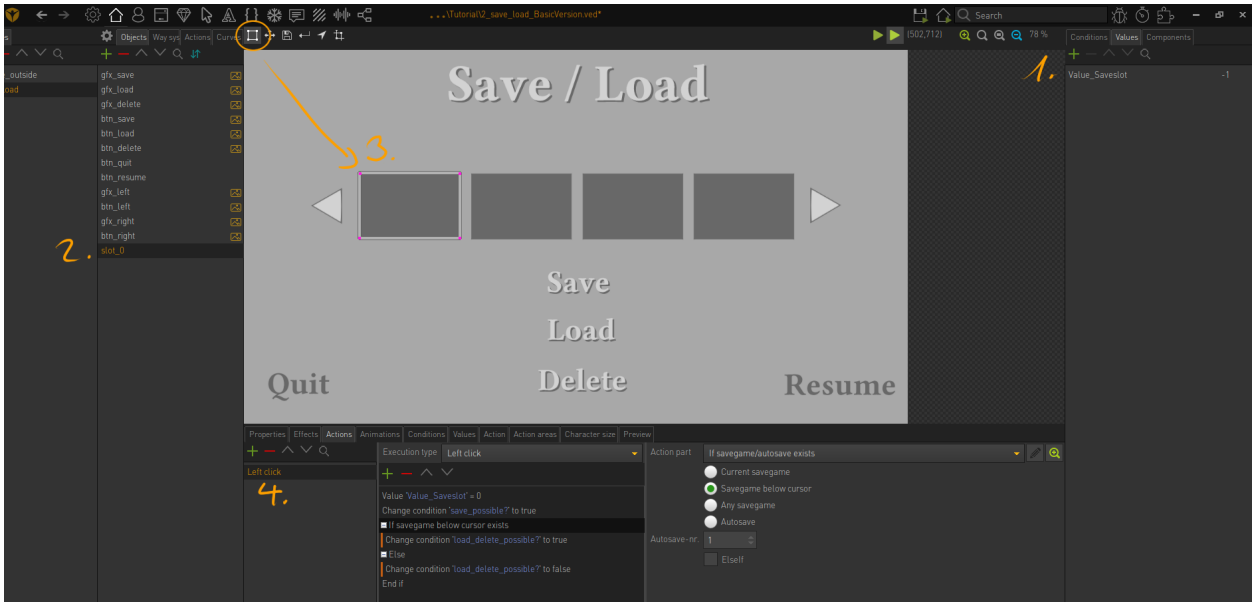


Fig. 2-9: Place an object on a save slot to have something to interact with

Fig. 2-9:

- 1. Create a value called "Value_Saveslot" and set it to -1. – This value will control which slot gets highlighted ("-1" means: none).
- 2. Create a new object and call it "slot_0".
- 3. Draw an object area for "slot_0" around the first save slot.
- 4. Add a "Left click" action for "slot_0" with the following setup of action parts:
  - Set the value "Value_Saveslot" to 0.
  - Change the condition "save_possible?" to true. – This will make the save button available.
  - Add an "If savegame exists" action part and select "Savegame below cursor". – This will check if the save slot we clicked on already holds a savegame (although we placed a clickable object on the slot's position, the engine still registers the slot under the cursor).
  - Change the condition "load_delete_possible?" to true. – This will make the load and delete buttons available, if the player selected a slot that holds a savegame.
  - "Else"
  - Change the condition "load_delete_possible?" to false. – This will hide the load and delete buttons, if the player clicks on an empty slot again.
  - "End if"
- Repeat steps 2 to 4 for the three remaining slots (I recommend duplicating and adjusting the "slot_0" object). In the first action part change "Value_Saveslot" according to the corresponding slot number (1 for "slot_1", …).

We start counting "Value_Saveslot" with 0 instead of 1, because the engine also starts counting the save slots with 0. This is not important in this basic version of the save menu, so you are free to use the numbers you like, but in a later version it's better to have the value match the internal numbering of the engine.

Now to the actual highlighting of the save slots (Fig. 2-10). As with the buttons highlighting before, we need another object to hold our highlight image:

- 1. Create a new object and call it "gfx_slot_0".
- 2. Add the highlight image ("menu_highlight.png") and place it on the first save slot.
- 3. Link the "Value_Saveslot" value to the object and make it validate for "= 0".

The highlight object will be visible, when "Value_Saveslot" is 0. The value gets changed to 0 when the player clicks on the "slot_0" object (see previous page), so this matches up.

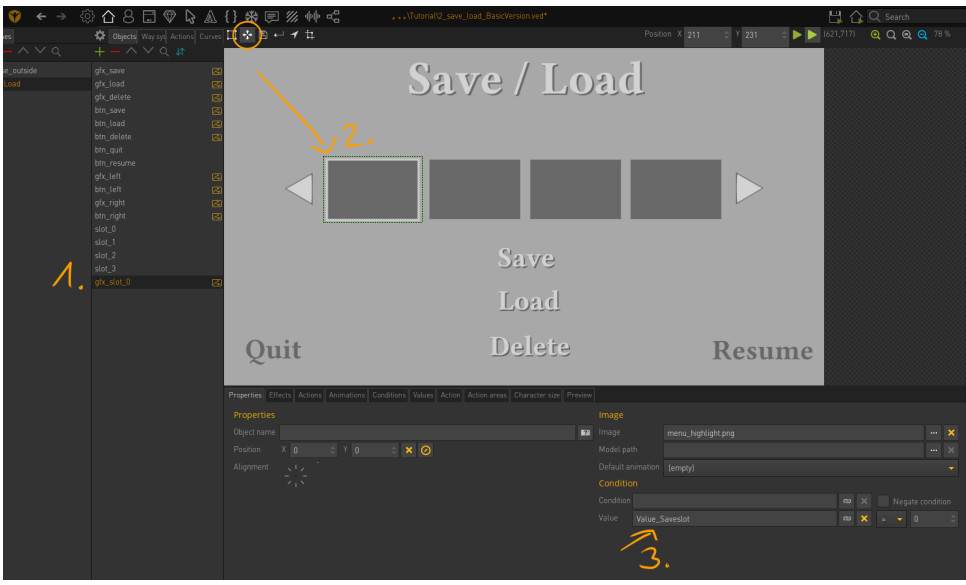


Fig. 2-10: Add a highlight object for a save slot

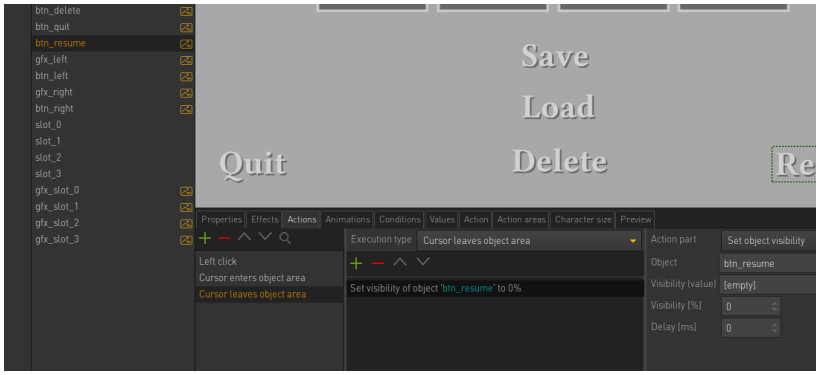


Fig. 2-11: Add hover highlighting for the quit and resume buttons

- Repeat steps 1 to 3 for the remaining save slots and name the objects "gfx_slot1" to "gfx_slot3" (again, duplicating the existing object will save you some time). Add the same highlight image to these objects, place them on the according slot and in step 3 set the number of the slot.
- In the "reset_button" action add the action part to set the "Value_Saveslot" value back to -1 (Fig. 2-12)

2.4 Enhance quit and resume buttons

Let's quickly add the active states of the quit and resume buttons (which we already created in the previous version).

We don't need to add a new object here, because our background already holds the inactive images for these buttons.

- Add the active images to "btn_quit" and "btn_resume" and create "Cursor enters object area" and "Cursor leaves object area" actions where you set "btn_quit" and "btn_resume" to 100 % and 0 %, respectively (Fig. 2-11).

The buttons will still work, even when their visibility is set to 0 %.

- Add these two objects to our list of buttons to reset the visibility (Fig. 2-12).

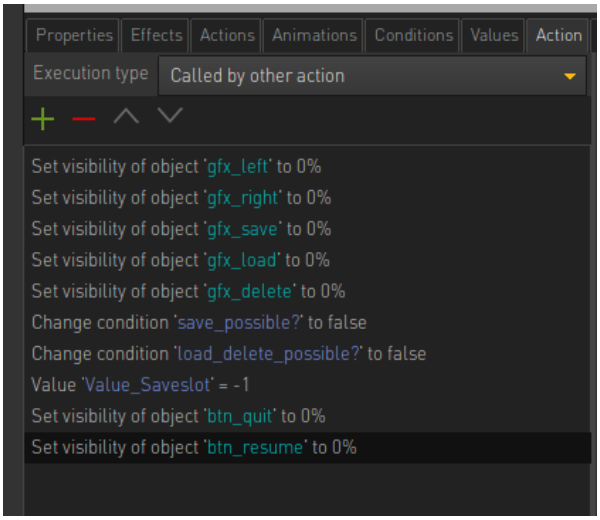


Fig. 2-12: Complete the reset action

3. Confirm It

Deleting or overwriting an existing savegame must not happen by accident. That's why in this version we add a confirmation dialog for both of these actions as well as for our quit button. Of course, you can also add the confirmation for the load button, but we skip that.

- 1. Add a new interface and name it "Confirmation" (Fig. 3-1).
- 2. In the interface properties, add the background image.
- 3. Assign the interface class "Secondary interface".
- 4. Since our background image has the size of our game, set the "Displacement" to "Absolute" and leave the "Offset" at [0,0] to center the image on the screen.

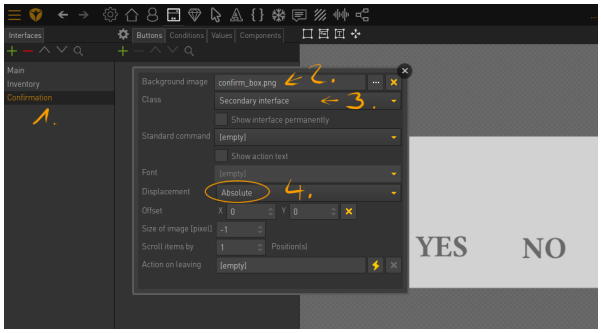


Fig. 3-1: Add an interface for the confirmation dialog

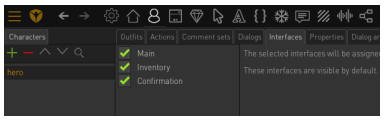


Fig. 3-2: Assign interface to the character

Three tasks

We will use the same confirmation dialog for handling all three tasks: overwrite, delete, and quit.

- Add a value "Selection_confirm" in the "Values" tab of the interface with an initial value of -1.

This value will store, which of the three tasks is the current one. We define:

0 = quit / 1 = delete / 2 = overwrite

Depending on that value, we'll show a different text on the dialog. Actually they're just images of text (Fig. 3-4).

- 1. Add a button called "Text_quit" to the "Buttons" list.
- 2. Make sure, the "Button type" is "Action area".
- 3. Link the "selection_confirm" value to the button and make it validate for "= 0".
- 4. Add the text image file as "Image (inactive)" and place it.
- Repeat steps 1 to 4 for the two other texts, with validation for "= 1" and "= 2", respectively.

- Assign the interface to our main character to be able to use it (Fig 3-2).

In the "Game properties" (Fig. 3-3):

- 1. Add the action part "Show/Hide interface" to the game's start action, choose the "Secondary Interface" class and select "Hide". That prevents the interface from being visible from the beginning.
- 2. Disable "Auto hide interfaces in menu", else our confirm dialog stays hidden.

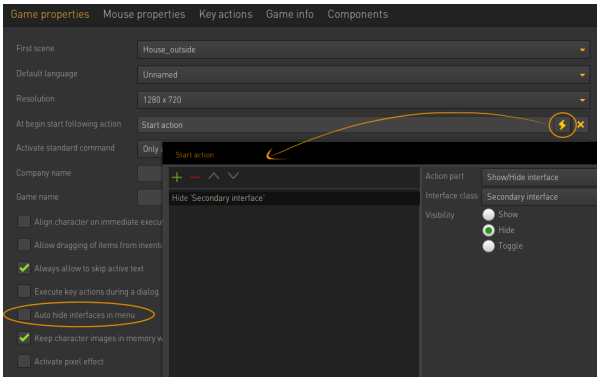


Fig. 3-3: Don't hide interfaces in menus, but hide the interface at game start

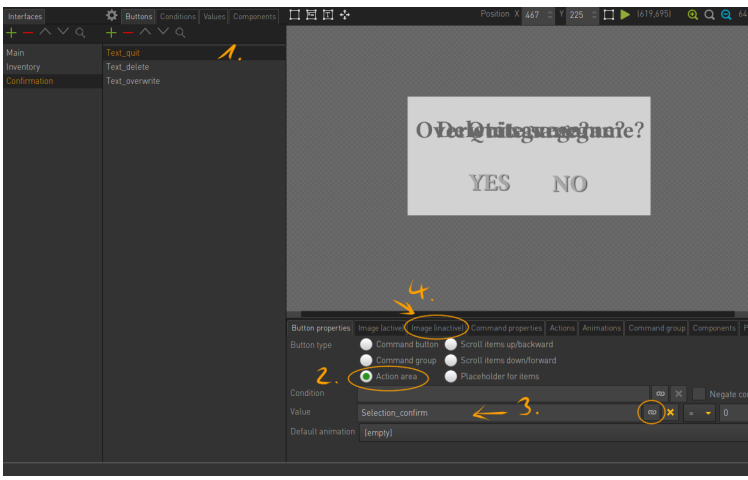


Fig. 3-4: Add text as images (placed on top of one another, but only one of them will show)

Yes and No buttons

The player's choice of "YES" or "NO" is painted onto the background image. We have to turn those words into buttons and make them work (Fig. 3-5).

- Add a new button called "btn_yes" and make sure it's an "Action area" type.
- Draw an object area around the word.
- Add a "Left click" action.

Inside this action, we execute commands depending on the "Selection_confirm" value:

- If "Selection_confirm" = 0
  - Quit game
- Else if "Selection_confirm" = 1
  - Delete the current savegame
- Else if "Selection_confirm" = 2
  - Save game
- End if

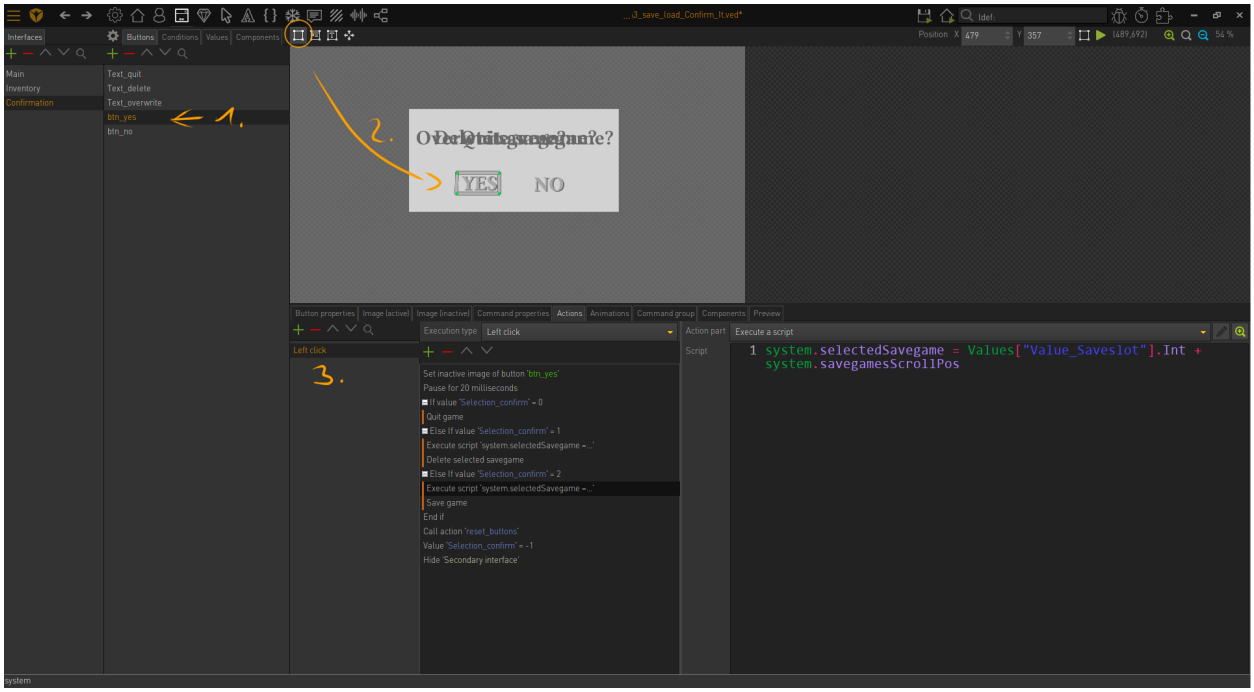


Fig. 3-5: Add the yes (confirm action) button

Additionally, we have to add a small piece of Lua code before deleting and saving. That's because the save slots (the rectangular areas we created with the floppy disk tool in chapter 1.2, not their graphical representation) get detected by the cursor even with our confirm interface visible. That means, the player may accidentally click on one of them after the confirm dialog opened, thus changing the selected slot without noticing it (because with the confirm interface overlaying the menu, our save slot highlighting wouldn't get triggered).

- Add two "Execute a script" action parts before the delete and save action parts, respectively, and enter the following:

```
system.selectedSavegame =
 Values["Value_Saveslot"].Int +
 system.savegamesScrollPos
```

This script sets the selected savegame to match the save slot the player chose.

- Likewise, add the execution of the following script to the "reset_buttons" action:

```
system.selectedSavegame = -1
```

- Now add a new button called "btn_no" and set it up just like the yes button (except for the action parts).
- Add the following three action parts to the "Left click" actions of the no button as well as of the yes button (after the if construct).

- Call the "Reset_buttons" action
- Reset "Selection_confirm" to -1
- Hide the "Secondary interface"

That will reset everything and close the dialog.

Like with all other buttons in this tutorial, we'll add highlighting for both the yes and no buttons. The good thing with interface buttons is, you can add two images to them: one for the inactive and one for the active state. In our case, the inactive state is already painted to the background though.

Do the following for both buttons:

- Add the button's active state image on the "Image (active)" tab.
- Place the image.
- Add an action for the cursor entering the button area and add the action part "Set active/inactive image" with the button linked and the "Active image" option selected.
- Do the same for the mouse leave action, but with selecting "Inactive image" (even if we didn't attach one).
- In the "Left click" action also add the "Set inactive image" action part (see Fig. 3-5).

Connect the confirm dialog

Back in the menu scene, we have to change the actions for the quit, delete, and save buttons we set up earlier, because we now want to call our confirm dialog.

- In the quit button's "Left click" action, delete the "Quit game" action part and add the following ones:
  - Set "Selection_confirm" to 0.
  - Show "Secondary interface".
- In the delete button's "Left click" action, delete the two action parts we added earlier and add the following ones:
  - Set "Selection_confirm" to 1.
  - Show "Secondary interface".

We don't have to delete the action parts from the save button, because saving the game doesn't change, unless it's overwriting an existing savegame. We need to add an "if" statement and some other action parts (Fig. 3-6):

- In the save button's "Left click" action, build the following setup of action parts (the red ones were already added before):
  - If selected savegame exists
    - Set "Selection_confirm" to 2.
    - Show "Secondary interface".
  - Else
    - Save game
    - Set "Selection_confirm" to -1.
    - Call "reset_buttons" action
  - End if

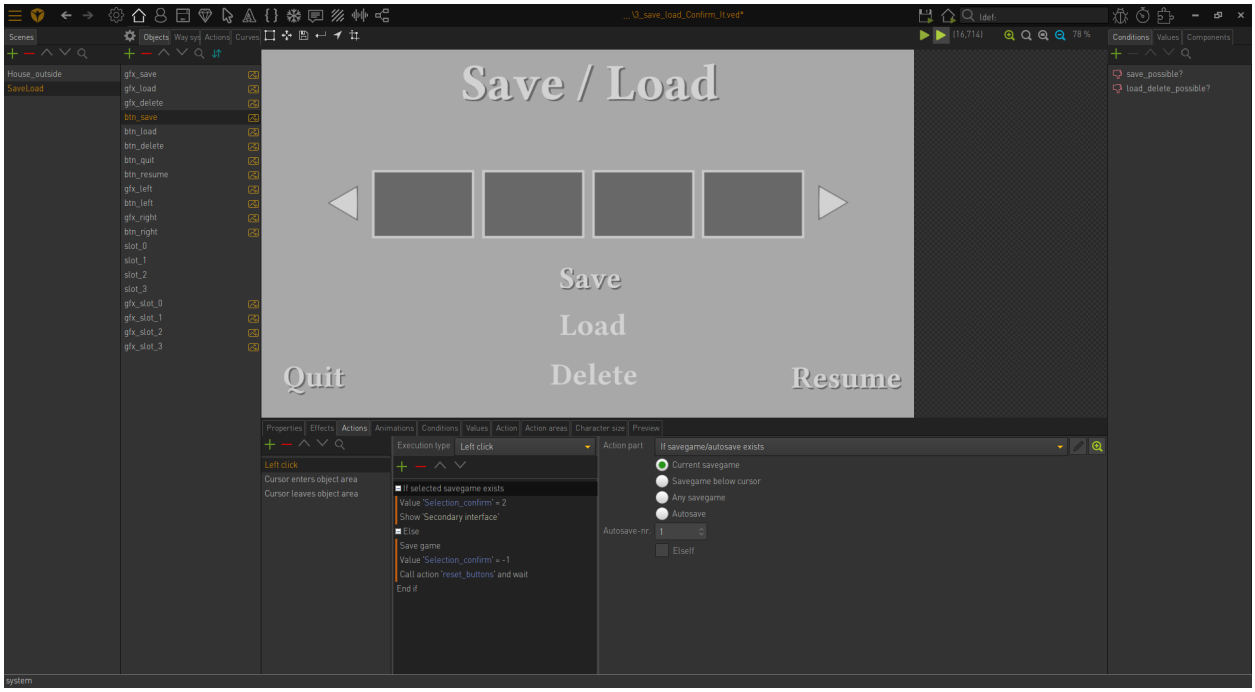


Fig. 3-6: Adjusting the save button's left click action.

4. Some Bells & Whistles

By now, our save/load menu is already working quite nicely. In this final version we try to slightly enhance the user experience.

4.1 A quicker save

Currently the user has to choose a save slot before being able to save the game. In most cases he'll pick the first free slot, so why not create a savegame in that position, if the player doesn't choose a slot at all but only hits the save button?

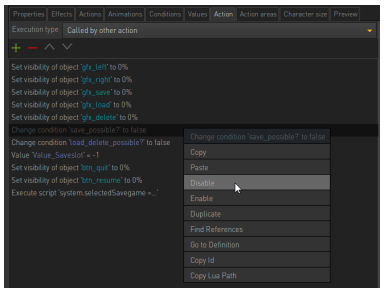


Fig. 4-1: Disable resetting a condition

The first thing to do is make the save button visible (and thus clickable) all the time.

- Change the initial state of the "save_possible?" condition to "True".
- In the "reset_buttons" action, deactivate the action part "Change condition 'save_possible?' to false" by selecting it, opening the context menu (right-click) and choosing "Disable" (Fig. 4-1).

You could as well just set the condition to "True" here or remove it completely from the project, if you don't want to use it at some point to prevent saving (maybe at game start). That depends on your particular game setup. We'll keep it for now, but don't use it – if only to teach you how to disable action parts. This possibility may come in handy especially when testing your game.

Now we extend the save button left click action. Currently we have an "if" construct to differentiate between overwriting an existing save slot and saving in an empty (selected) one. Saving in the first empty slot will become the third option (Fig. 4-2).

- Add an "Else If value 'value saveslot' = -1" action part. – That means, if no slot has been chosen.
- Add a "Save game" action part and one to call the "reset_buttons" action, like we did for saving in an empty slot.
- Add an "Execute a script" action part before the save action part and enter the following:

```
system.selectedSavegame = system.savegamesCount
```

This selects the first empty save slot (remember that slot numbering starts with 0).

- Add another "Execute a script" action part after the save action part and enter the following:

```
local n = system.savegamesCount -
system.savegamesScrollPos -3

Values["move_slots"].Int = math.abs(n)

if n > 0 then
 startAction(Actions["savegame_right"])
elseif n < 0 then
 startAction(Actions["savegame_left"])
end
```

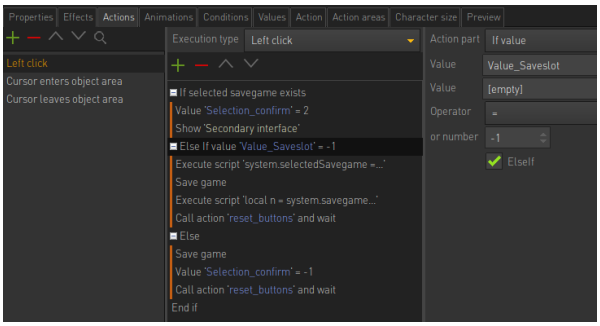


Fig. 4-2: Extend the save button action

This script will scroll through the save slots, until the one we just saved in is visible. For example: if four slots are already occupied and the player saves (without selecting a slot first), the new savegame will go into slot five, which may be out of sight.

First we calculate, how many times we have to (virtually) hit that left/right scroll arrow button. The "-3" works for our 4 slot menu with a scrolling amount of 1 per click. If your setup is different, you need to adjust the calculation. The absolute value gets stored in a Visionaire value called "move_slots". And depending on the calculated number being positive or negative, we call an action that will scroll to the right or to the left.

- Add a value called "move_slots" on the menu's "Values" tab and set the initial value to 0.

- Add a another value called "loops" and set the initial value to 0. – This will serve as a counting variable.
- On the menu's "Actions" tab, add a new action called "savegame_right". – This action gets called through the above script, when we need to scroll forward.
- Add the following setup of action parts to "savegame_right" (Fig. 4-3):
  - Value 'loops' = 0
  - If value 'loops' != 'move_slots'
  - Value 'loops' + 1
  - Show next savegames
  - Jump 3 action parts backwards (Jump relative, -3)
  - End if
  - Value 'move_slots' = 0

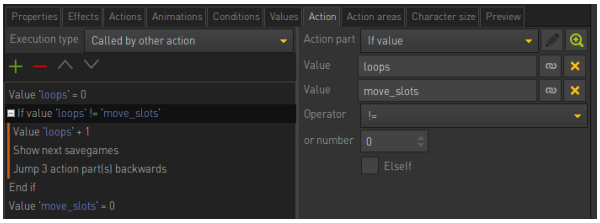


Fig. 4-3: Create a loop to scroll through the save slots step by step

This creates a loop that scrolls the savegames by the amount stored in the "move_slots" value.

- Duplicate the action, rename it "savegame_left" and change the third action part to show the previous instead of the next savegames.

The scrolling also makes sense when deleting a savegame, so we'll add it there, too:

- Copy the second "Execute a script" action part from the save action and paste it into the left click action of the "btn_yes" button of the confirmation dialog, just after "Delete selected savegame" (Fig. 4-4).

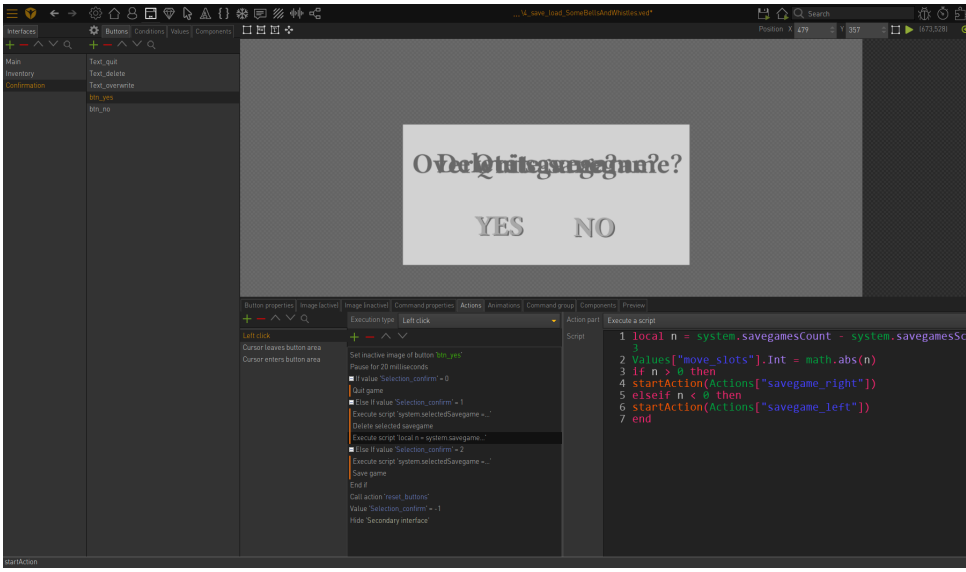


Fig. 4-4: Add the scroll script to the delete savegame action

4.2 Better slot highlighting

Prevent scrolling confusion

What if the player decides to select a save slot und then – with the slot highlighted – starts to scroll? The slots will scroll of course, but the highlight graphic will stay where it is, because there is no real connection between this graphic and a particular slot. Remember we just added objects to place the graphics upon the slots. Thus we will end up having a save slot highlighted that is not the (internally stored) selected one.

Therefor we need to hide the highlighting and reset the chosen slot whenever one of the scroll button arrows is clicked. Additionally, we'll disable the scroll buttons once we have reached the end of the savegames list in either direction.

- Add a new "Called by other action" action and name it "reset_slot".
- Move the following three action parts from the "reset_buttons" action to the new one (Fig. 4-5):
  - Change condition 'load_delete_possible?' to false
  - Value 'Value_Saveslot' = -1
  - Execute script (where SelectedSavegame is set to -1)
- In the "reset_buttons" action, add an action part to call the "reset_slot" action.

We have just "outsourced" three action parts from our original reset action, so we can call them independently through the "reset_slot" action. The "reset_buttons" action did effectively not change.

► Change the left click action of the "btn_left" button as follows (Fig. 4-6):

- If lua result

```
return system.savegamesScrollPos ~= 0
```

– This ensures, that we're not already at the start of the savegame list. If we are, do nothing.

- Show previous savegames
- If lua result

```
return system.savegamesScrollPos == 0
```

– This checks, if we're at the start of the savegame list, AFTER we scrolled to the left. If so, we hide the arrow highlighting to indicate that.

- Set visibility of "gfx_left" to 0 %
- End if
- End if
- Call action "reset_slot"

The check, whether we have reached the start of the list, is also required in our arrow button highlighting action (Fig. 4-7).

► In the "Cursor enters object area" action of "btn_left", wrap the visibility action part in an "if" construct:

- If lua result

```
return system.savegamesScrollPos ~= 0
```

- Set visibility of "gfx_left" to 100 %
- End if

Now we have to make the same adjustments for the right scrolling button. The difference (besides the already existing "Show next savegames" instead of showing the previous ones) is the pieces of Lua code. We now have to check, whether we have reached the end of the list (instead of the start).

► Repeat the above steps for "btn_right" but with the following code changes:

```
return system.savegamesScrollPos ~=
system.savegamesCount
```

```
return system.savegamesScrollPos ==
system.savegamesCount
```

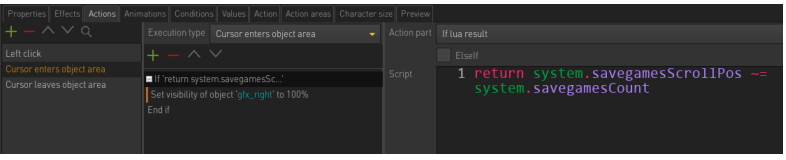


Fig. 4-7: Disable the arrow highlighting when reaching the end of the savegames list

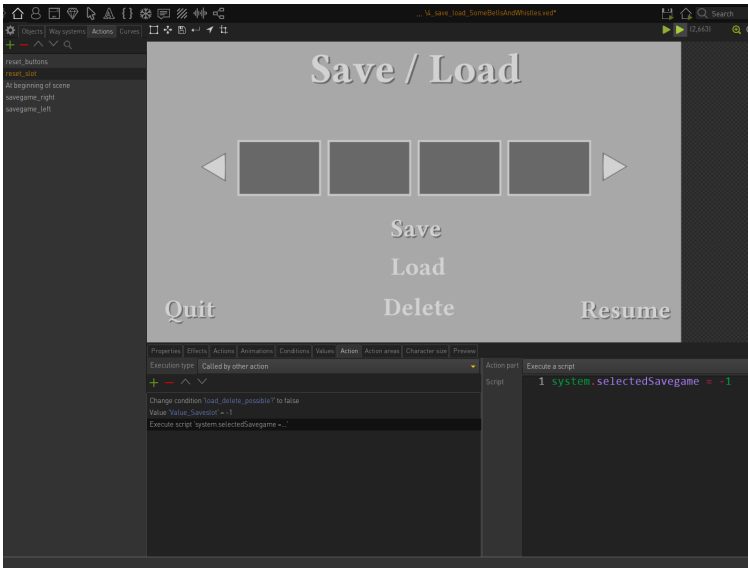


Fig. 4-5: Add a second "reset" function

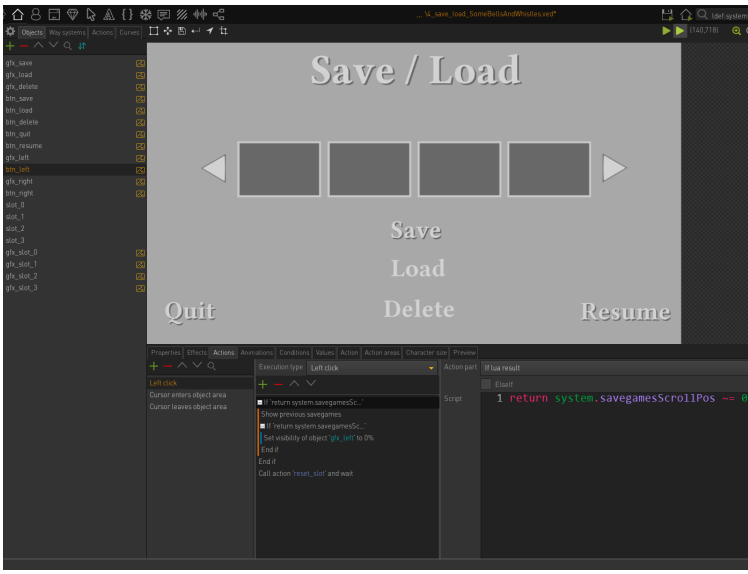


Fig. 4-6: Change the scroll button action

Prevent highlighting of unavailable slots

The last enhancement we add to our save/load menu is related to the "problem" that Visionaire does not allow for gaps in our savegames list, i. e. empty save slots in between occupied ones. Actually, this behaviour is not a real problem but makes complete sense. Gaps are expendable.

In our current menu it becomes a problem though, because the player may highlight every slot he wants – but the engine might save in another one instead. That's why we'll automatically highlight the first available slot, if the player selects an inappropriate one.

Since it isn't possible to leave a gap when selecting the first save slot, we don't have to change anything for the "slot_0" object.

► Change the left click actions for "slot_1", "slot_2", and "slot_3" as follows (Fig. 4-8):

- Move the "Set value 'Value_Saveslot'" action part from the top to below "If savegame below cursor exists".
- Add an "Execute a script" action part below "Else" with the following Lua code:

```
local pos = system.savegamesCount -
system.savegamesScrollPos
Values.Value_Saveslot.Int = pos
```

Remember that our "Value_Saveslot" value controls the slot highlighting. In case the player wants to overwrite an existing savegame, we can set the slot number they selected, just as we did before.

If the player selects an empty slot ("Else"), we set the number of the first empty slot. So whatever slot is selected, the first empty slot gets highlighted, because that's where the engine will put it.

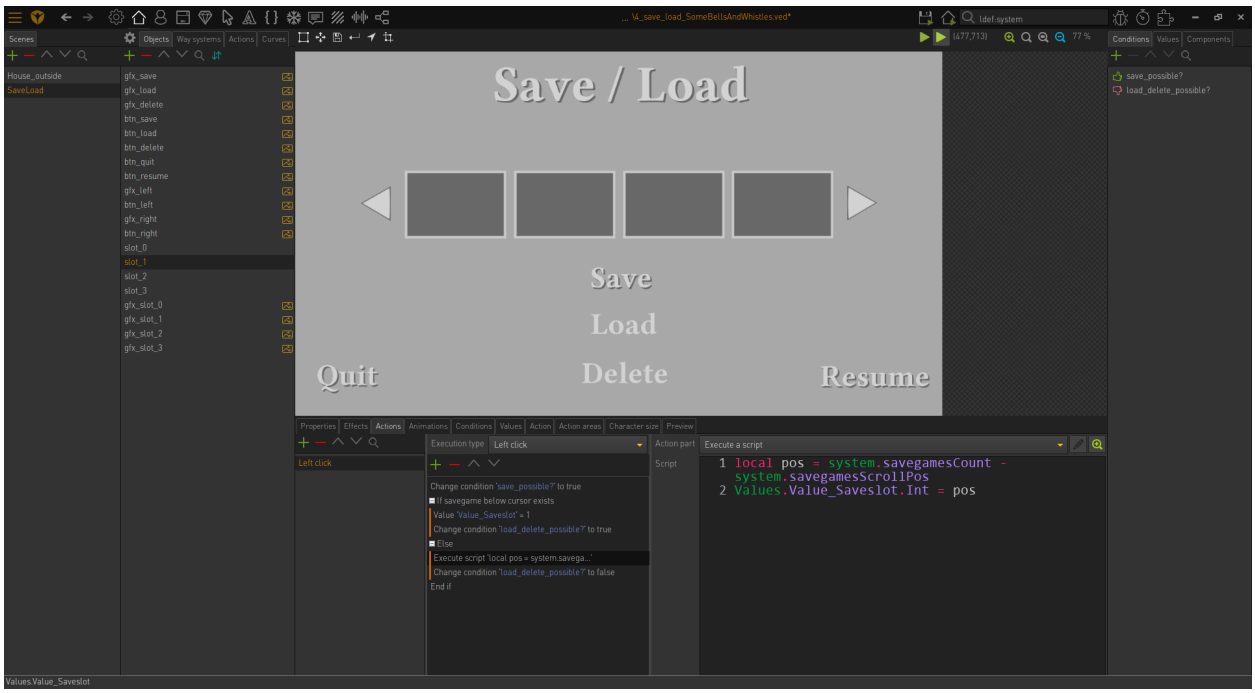


Fig. 4-8: Modify the slot highlighting

Save/Load Menu created by Esmeralda

Tutorial written by Esmeralda & Einzelkämpfer, October 2022