9-verb MI-style template v1.3

Contents

Introduction

9-verb MI-Style is a template that allows you to recreate those classic SCUMM games.

If you like to create a game with this template, you have to re-think some concepts, you usually use when creating a game with AGS.

The two new concepts are: event handling and default actions, using extensions.

Let's take a look at the event handling, or in other words: stuff that happens after you have clicked on something. Normally you create a function for each event of an object, a hotspot etc. Functions like cup_Interact() or cup_Look(). Using this template you only need one the any click event. Inside these functions you distinguish between the different interactions. So a typical any_click function looks like this:

```
function cup_AnyClick()
{
 // LOOK AT
 if(UsedAction(eGA_LookAt)) {
 player.Say("It's a blue cup.");
 }
 // USE
 else if(UsedAction(eGA_Use)) {
 player.Say("I'd rather pick it up.");
 }
 // PICKUP
 else if(UsedAction(eGA_PickUp)) {
 player.Say("Okay.");
 any_click_walk_look_pick(108, 100, eDir_Up, "You are now mine.",oCup.ID,
iCup);
 }
 //USE INV
 else if(UsedAction(eGA_UseInv)) {
 Unhandled();
 }
 // don't forget this
 else Unhandled();
}
```

The function "any_click_walk_look_pick" is explained in the function reference.

So you see, everything is inside a single function, instead 4 seperate functions. Also instead of checking the cursormodes, the function UsedAction is called to determin the event/action. The current defined actions are:

```
eGA_LookAt, eGA_TalkTo, eGA_GiveTo, eGA_PickUp, eGA_Use, eGA_Open, eGA_Close,
eGA_Push, eGA_Pull,eGA_UseInv, eMA_Default and eMA_WalkTo
```

For inventory items, it's a little bit different, because there is no any_click event in the room editor. So you first start with "other click on inventory item", which creates the function iCup_OtherClick (in case you have an item, called iCup).
Now copy this function name and paste it at other events, like Interact, look, talk and so on. In the end, you only have one function name in all five events. You can also take a look at the sample items.

The second main aspect of the GUI are the extensions. You add an extension to a location (Hotspots, Objects etc.) by editing its description.

For example, let's take an object. In the sample room, the object is called oCup and the description is simply "Cup".

When move the cursor over this cup and no extension is defined, the default action will be "look at". Also the corresponding verb button in the gui starts to highlight. Now we can change this behaviour with adding an angeled bracket, followed by one of the following letters:

```
n: nothing / default
g: give to
p: pick up
u: use
o: open
l: look at
s: push
c: close
t: talk to
y: pull
v: variable extension
e: exit
```

Let's give oCup the description "Cup>p". Now the right-click action has changed. If you now move the mouse on the cup, the verb button "Pick Up" is highlighted. If you right-click the object, the any_click function from above is called. It checks for the used action and will perform the chosen action.

Extensions are also explained in the function reference.

The last thing you should know about, is the global variable "ItemGiven". If you like to give an item to a character, use this variable to check, which item has been given. For example:

```
if (UsedAction(eGA_GiveTo))
{
  if (ItemGiven == iCup)
  {
    player.Say("Do you want this Cup?");
    cBman.Say("No, thank you.");
  }
  else if (ItemGiven == iKey)
  {
    player.Say("Is that your key?");
    cBman.Say("Of course. You have it from me.");
  }
  else Unhandled();
}
else Unhandled();
```

If you need to use "ItemGiven" in other scripts than the global one, you need to import it manually. It's not defined in the Global Variable Editor of AGS.

Using Inventory Items

There are currently three ways of using an inventory item, you can choose from.

1. „Use“ only
   For this, you need to add the use-extention „>u“ to the description of the item and an event function for „Interact inventory item“. If you have followed the instructions in this manual you probably already have it there. This option might come handy for a watch. Clicking on it always gives you the current time. You can not give it away or use it with different items.
2. „Use“ and „Use with“
   Here you need to remove the use-extension from the description, but still keep the event function. This allows the player to directly use the item by clicking on the „use“-verb first. Directly clicking the items results in „use-with“. Sticking to the watch-example: using the watch with the verb-button sets an alarm. Clicking directly on it in the inventory results in „use with“, so you can use the watch with a shelf to hide it there. But please note that it might be hard for the player to understand, that using the verb button and using the inventory directly are two different things.
3. „Use with“ only
   For the last option, you need to remove the use-extension and remove the event function. Yep, that's right: on the right side of „Interact inventory item“ is no function at all. If you then use the item, whether it's via the

verb-button or a direct click, the action always stays „use with“.

Exit Extensions

As of version 1.1 you can add an exit extension to hotspots and objects. Clicking on such a hotspot will make the player walk to it and change the room afterwards. There are several advantages compared to the usual methods like 'screen edges' or stand-on hotspot functions:

- works with objects and vertical hotspots (like cave entrances)
- supports double click to skip the walking
- optional walking off the screen: if you set the exit hotspot towards a screen edge, you can make the player leave the screen and change the room after that.

This is how it works:

First of all create your hotspot and let it have the '>e' extension. Now switch over to the events (that little flash) and add the Usermode_1 hotspot event. Eventually you'll end in the room script with a function called 'hExit_Mode8'.

In that function, all you have to do is to script the room change. e.g.

```
player.EnterRoom(1, 76, 111, eDir_Right, true);
```

This function is almost similar to the AGS function player.ChangeRoom, you can look it up in the function reference below.

If you want the player to leave the screen, you have to change the extension of the hotspot. These ones are possible:

```
el: left
er: right
eu: up
ed: down
```

If you have an exit on the right side of your screen and want the player to leave the screen on that side, your hotspot description should be called:

```
Exit>er
```

Now the character will walk to the clicked location and keeps on walking for another 30 extra pixels.

That offset can be changed in the script header.

If you simply call your hotspot:

```
Exit>e
```

No additional walking will occur. This is useful for exits not being at the screen border. There's also an example in the second room of the demo template.

Language & Translation

Currently (v 1.3) the GUI supports German, French and Spanish. If you like to help translating this template, please drop me a PM at the AGS Forums.

If you like to create your game in a different language than english, you need to set it up. At guiscript.asc you'll find the line:

```
int lang = eLangEN;
```

At the time of writing, valid values are: eLangEN, eLangES, eLangFR and eLangDE. Setting this variable to one of these values will translate all your GUIs, including all provided dialogs. The unhandled events will stay unchanged however. Those are still needed to be changed directly.

To switch the language in a .trs translation file, tell your translators to look out for the line.

```
GUI_LANGUAGE
```

Now simply translate that line with DE, EN, ES or FR to set the GUI to the corresponding language.

Function Reference

The functions in this section are only available if you have created your game using that template.

Math and Helper Functions

int Absolute(int value);

int Offset(int point1, int point2);

int getButtonAction(int action);

function disable_gui();

function enable_gui();

bool is_gui_disabled();

Absolute

```
int Absolute(int value);
```

Returns the absolute value of a given value.

Offset

```
int Offset(int point1, int point2);
```

Returns the offset between to two given values.

getButtonAction

```
int getButtonAction(int action);
```

Returns the connected action of a verb button. The actions for the verb buttons are not "hard-wired" inside the GUI-script, but defined in the function SetButtonAction.

See also: SetActionButtons, AdjustLanguage

disable_gui

```
function disable_gui();
```

This functions disables the GUI and hides it.

See also: is_gui_disabled, enable_gui

enable_gui

```
function enable_gui();
```

This function enables the GUI again.

See also: is_gui_disabled, disable_gui

is_gui_disabled

```
bool is_gui_disabled();
```

Returns true, if the GUI is currently disabled, false otherwise

See also: is_gui_disabled, disable_gui

GlobalCondition

```
int GlobalCondition(int parameter);
```

Used to check for conditions that are used many times in the script. For example, it's used to check, if the mouse cursor is in the inventory and the mode walk or pickup are selected.
Returns 1, if the condition is true and 0 otherwise.

GetLucasSavegameListBox

```
function GetLucasSavegameListBox(ListBox*lb);
```

This is a helper function to initialize the save and restore dialogs.

Action Functions

function UsedAction (Action test_action);

bool isAction(Action test_action);

function SetActionButtons(Action action, String button_definition);

function SetDefaultAction(Action def_action);

function SetAction(Action new_action);

function SetAlternativeAction(char extension, Action alt_action);

function CheckDefaultAction();

These functions are mainly used to control the verb buttons.

UsedAction

```
function UsedAction (Action test_action);
```

Used to determine, which action has been selected by the player. Instead of checking cursormodes, this function is used.

isAction

```
bool isAction(Action test_action);
```

Used to check, if the current action is the one, given in the parameter.

SetActionButtons

```
function SetActionButtons(Action action, String button_definition);
```

This functions connects the verb buttons with the action and is also used to assign / change the graphics of the verb buttons.

See also: AdjustLanguage

SetDefaultAction

```
function SetDefaultAction(Action def_action);
```

Used to define, which action is being used, if no verb has been clicked. Usually this is "walk to".

SetAction

```
function SetAction(Action new_action);
```

Since the cursormodes are bypassed, this function defines the current action. Among other things, this function is called by clicking a verb button.

SetAlternativeAction

```
function SetAlternativeAction(char extension, Action alt_action);
```

This function makes the right-click shortcuts work. If you use extensions like ">p" (e.g. pickup), this function makes sure, that the correct verb button is highlighted.

See also: CheckDefaultAction

CheckDefaultAction

```
function CheckDefaultAction();
```

This function checks for a given extension in hotspots, objects and characters. If there isn't an extension, a default action is given, e.g. "Talk to" if the mouse is over a character. In case of a given extension, the default actions are being overridden. It is also defined here, which letters are causing what default action. See the chapter Extensions for more details.

See also: Extensions

UpdateActionBar

```
function UpdateActionBar();
```

This function is used to show and update the status bar. It checks for an extension, triggers the translation and renders the results on screen.

See also: TranslateAction, RemoveExtension

Player Functions

function freeze_player();

function unfreeze_player();

*function SetPlayer(Character*ch);*

function FaceDirection (this Character, eDirection dir);*

function EnterRoom(this Character, int newRoom, int x, int y, eDirection dir, bool onWalkable);*

freeze_player

```
function freeze_player();
```

Use this function to prevent the player from moving by the following movement functions of the template.

See also: unfreeze_player

unfreeze_player

```
function unfreeze_player();
```

Use this function to undo the freeze_player function and let the characters move again.

See also: freeze_player

SetPlayer

```
function SetPlayer(Character*ch);
```

```
Usage: cEgo.SetPlayer();
```

Similar to the AGS function Character.SetAsPlayer(). The difference is, that make the previous character clickable again, whereas the new character gets unclickable.

FaceDirection

```
function FaceDirection (this Character*, eDirection dir);
Usage: cEgo.FaceDirection(eDir_Left);
```

Similar to the AGS function Character.FaceLocation. Possible directions are:

```
eDir_Left, eDir_Right, eDir_Up, eDir_Down
```

EnterRoom

```
function EnterRoom(this Character*, int newRoom, int x, int y, eDirection
dir, bool onWalkable);
Usage: cEgo.EnterRoom(1,15,15,eDir_Left,true);
```

Similar to the AGS function Character.ChangeRoom. The difference is, that you can also define, it which direction the character should look.

Using this function makes the character turn to the direction, mentioned above.

Cancelable, semi-blocking move-player-character functions

int MovePlayer(int x, int y);

int MovePlayerEx(int x, int y, WalkWhere direct);

*int GoToCharacterEx(Character*chwhogoes, Character*ch, eDirection dir, int xoffset, int yoffset, bool NPCfacesplayer, int blocking);*

*int GoToCharacter(Character*charid, eDirection dir, bool NPCfacesplayer, int blocking);*

*int NPCGoToCharacter(Character*charidwhogoes, Character*charidtogoto, eDirection dir, bool NPCfacesplayer, int blocking);*

int any_click_move(int x, int y, eDirection dir);

int any_click_walk(int x, int y, eDirection dir);

int any_click_walk_look(int x, int y, eDirection dir, String lookat);

*int any_click_walk_look_pick(int x, int y, eDirection dir, String lookat, int objectID, InventoryItem*item, AudioClip *sound=false);*

*int any_click_use_inv (InventoryItem*item, int x, int y, eDirection dir);*

function GoTo(int blocking);

function Go();

function WalkOffScreen();

Semi-blocking means, that you can cancel the movement, but certain code is only executed, after the character has actually reached it's goal. To achive this, these functions are called inside an if-clause.

```
e.g.: if(MovePlayer(20,20)) Display("The player has reached the
destination.");
```

If the player's character reaches the coordinates 20,20, the message "I'm there" is being displayed. If the movement is being cancelled by a mouseclick, the message doesn't appear.

MovePlayer

```
int MovePlayer(int x, int y);
```

Moves the player character around on walkable areas, a wrapper for MovePlayerEx.

Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before.

See also: MovePlayerEx

MovePlayerEx

```
int MovePlayerEx(int x, int y, WalkWhere direct);
```

Move the player character to x,y coords, waiting until he/she gets there, but allowing to cancel the action by pressing a mouse button.

Returns 1, if the character hasn't cancelled the movement and 0 if the movement has been cancelled before. 2 is returned, if the characters has actually reached it's goal: eg. if a walkable area is being removed while the player is still moving.

GoToCharacterEx

```
int GoToCharacterEx(Character*chwhogoes, Character*ch, eDirection dir, int
xoffset, int yoffset, bool NPCfacesplayer, int blocking);
```

Goes to a character staying at the side defined by 'direction': 1 up, 2 right, 3 down, 4 left and it stays at xoffset or yofsset from the character. NPCfacesplayer self-explained.

blocking: 0=non-blocking; 1=blocking; 2=semi-blocking

Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before.

See also: GoToCharacter, NPCGoToCharacter

GoToCharacter

```
int GoToCharacter(Character*charid, eDirection dir, bool NPCfacesplayer, int
blocking);
```

The same as GoToCharacterEx, just with the one character being the player and a default offset of x=35px and y=20px.

Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before.

See also: GoToCharacterEx

NPCGoToCharacter

```
int NPCGoToCharacter(Character*charidwhogoes, Character*charidtogoto,
eDirection dir, bool NPCfacesplayer, int blocking);
```

The same as GoToCharacterEx, just with an default offset of x=35 and y=20

Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before.

See also: GoToCharacterEx

any_click_move

```
int any_click_move(int x, int y, eDirection dir);
```

Moves the player character to the coordinates given in the parameters. If the player reaches the destination, it's turns to the given direction.

Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before. You can use this kind of functions (including the movePlayer function which is called by this function), to check if the player actually reached

it's destination. For example:

```
if (any_click_move(130,110,eDir_Left)==1) player.Say("I've reached the
place.");
```

So the Message is only displayed, if the movement hasn't been cancelled.

See also: MovePlayer, MovePlayerEx

any_click_walk

```
int any_click_walk(int x, int y, eDirection dir);
```

This function is almost similar to any_click_move. But it's only called, if the current action is eMA_WalkTo.

See also: MovePlayer, MovePlayerEx, any_click_move

any_click_walk_look

```
int any_click_walk_look(int x, int y, eDirection dir, String lookat);
```

This function moves the player character to the given location, turns it to the given direction and lets it say the message, given in the string.

See also: any_click_walk

any_click_walk_look_pick

```
int any_click_walk_look_pick(int x, int y, eDirection dir, String lookat, int
objectID, InventoryItem*item, AudioClip *sound);
```

This function starts the same as any_click_walk_look. If an object ID > 0 has been given, this object is set invisible. Afterwards the inventory item is going to be added to the player's inventory and if there's an audioclip in the parameters, that one is played too.

The function return 0 if the action has been cancelled, before the player has reached the coordinates. 1 is returned if the player has reached the given destination, but has not picked up the item. 2 is returned, if the item has been picked up.

See also: any_click_walk_look, any_click_walk

any_click_use_inv

```
int any_click_use_inv (InventoryItem*item, int x, int y, eDirection dir);
```

This function moves the player to the given destination. It returns 0, if the action is unhandled, 1 is returned, if the action is handled, but has been cancelled. 2 is returned, if everything went fine. A possible usage is:

```
if (any_click_use_inv (iWrench,100,130,eDir_Left)==2) player.Say("I will now
repair this pipe.");
```

See also: any_click_walk_look, any_click_walk

GoTo

```
GoTo(int blocking);
```

Go to whatever the player clicked on. This function is used to intercept a walk-to event and check if the player has reached it's goal. E.g. this is used in the exit extension processing.
blocking: 0=non-blocking; 1=blocking; 2=semi-blocking

See also: Go, MovePlayer, WalkOffScreen

Go

```
Go();
```

Go to whatever the player clicked on. You can cancel the action, and returns 1 if the player has gone to it. This is just a shortcut for GoTo(2), which means semi-blockable.

See also: GoTo

WalkOffScreen

```
WalkOffScreen();
```

Handles the action of hotspots or objects with the exit extension ('>e'). Take a look at chapter about extensions to see what this function does.

See also: extension chapter

Door functions

function set_door_state(int door_id, int value);

int get_door_state(int door_id);

function init_object(int door_id, int act_object);

function set_door_strings(String lookat='0', String islocked='0', String wrongitem='0', String closefirst='0', String unlock = '0', String relock='0');

String get_door_strings(String what_type);

int any_click_on_door(int door_id, int act_object, int x, int y, eDirection dir, int nr_room, int nr_x, int nr_y, eDirection nr_dir);

*int any_click_on_door_special (int door_id, int act_object, int x, int y, eDirection dir, int nr_room, int nr_x, int nr_y, eDirection nr_dir, AudioClip *opensound, AudioClip *closesound, int key, int closevalue);*

This template implements a clever door scripting system, which is a real timesaver if you use a lot of doors. It uses a hotspot for the closed door and a non-clickable object, to show the opened door.

If you enter a room the first time, you have to set up its containing doors:

```
function room_FirstLoad()
{
 // Lock door on startup when entering the room
 set_door_state(20, 2);
 init_object(20, oDoor.ID);
}
```

This will set up a door with the id 20 to the state 2, locked. With "init_object", you connect the object, displaying a sprite of an opened door, with the door ID.

Now let's take a look at your hotspot's function:

```
function hDoor_AnyClick()
{
 if (any_click_on_door(20, oDoor.ID, 61, 104, eDir_Left, 1, 115, 135,
eDir_Down)==0) Unhandled();
}
```

This function is explained in detail later in this document. But for starters, this is all you have to do in the room script. And looks much harder than it is, just take a look at the sample room, supplied with this template.

If you want to have the script to show the correct default actions, you also need to change a line in the gui-script: so look for a function, called VariableExtensions.

In VariableExtensions, look at this line:

```
if (r==1 && h == 1)  OpenCloseExtension (20);
```

This tells the script, that Room 1 contains a hotspot with the id 1. This is connected to a door a door with the id 20. So now, the right-click should suggest open/close, depending on the door's state.

set_door_state

```
function set_door_state(int door_id, int value);
```

A door can have 3 different states:

```
0 = The door is closed
1 = The door is open
2 = The door is closed and locked
```

Call this functio to set a door state for the given door_id.

See also: get_door_state, init_object

get_door_state

```
int get_door_state(int door_id);
```

Returns the current state of a door.

```
0 = The door is closed
1 = The door is open
2 = The door is closed and locked
```

See also: set_door_state, init_object

init_object

```
function init_object(int door_id, int act_object);
```

Used to set up the corresponding object, used by the door with the given id. If the state of the door is closed, the object will be invisible. Otherwise, the object will be shown. The object stays unclickable all the time.

See also: set_door_state

set_door_strings

```
function set_door_strings(String lookat, String islocked, String wrongitem,
String closefirst, String unlock, String relock);
```

Use this function to define the messages, the player character says, when approaching a door.

```
lookat: shown, if the player looks at the door
islocked: shown, if the player tries to open a locked door
wrongitem: shown, it the player tries to unlock the door with a wrong item
closefirst: shown, if the player wants to lock a opened door
relock: shown, if the player locks a previously unlocked door
```

See also: get_door_strings

get_door_strings

```
String get_door_strings(String what_type);
```

Returns the message, which has been set up by set_door_strings. Accepted parameters are:

```
lookat, islocked, wrongitem, closefirst, relock
```

Remember, that these are strings, so don't forget the quotation marks.

```
Usage: if(!String.IsNullOrEmpty(get_door_strings("islocked")))
player.Say(get_door_strings("islocked"));
```

See also: set_door_strings

any_click_on_door

```
int any_click_on_door(int door_id, int act_object, int x, int y, eDirection
dir, int nr_room, int nr_x, int nr_y, eDirection nr_dir);
```

This function is used in the room script in combination with the door hotspot.
Parameters:

```
door_id: The door id, you have defined
act_object: The object, containing the open sprite
x,y: the walk-to point of the door (please don't use the built in "walk-to
coordinates" feature of the room editor.
dir: the direction, the player's character should face, after it reached x,y
nr_room: if the door is opened and walking through it, the player is being
send to this room
nr_x,nr_y: the x,y coordinates of inside of the new room
nr_dir: after the room change, the player faces this direction
```

This is the main function of the door scripts. With this you connect the hotspot with the door and the player's action. If you have defined default door sounds, these are also being called in this function. Also you can't unlock a door with this function. You need any_click_on_door_special for that.

See also: any_click_on_door_special

any_click_on_door_special

```
int any_click_on_door_special (int door_id, int act_object, int x, int y,
eDirection dir, int nr_room, int nr_x, int nr_y, eDirection nr_dir, AudioClip
*opensound, AudioClip *closesound, int key, int closevalue);
```

This function extends any_click_door with the following parameters:

```
opensound: custom sound to be played, when the door is being opend
closesound:  custom sound to be played, when the door is being closed
key: the id of the inventory item, that can unlock the door, -1 masterkey, -2
if the door cannot be unlocked
closevalue: default 0 (closed), but you can also set 2 (locked).
```

See also: any_click_on_door

Unhandled Events

In order to give a the player a feedback for actions the gamescreator hasn't thought of, unhandled events come into play. With a single function, you can achieve something like "That doesn't work" or "I can't pull that", which makes a game much more authentic and alive.

Unhandled

```
function Unhandled(int door_script);
```

Use this function at the end of your any_click functions in order to cause default reactions. For example:

```
function cChar_AnyClick()
{
if (UsedAction(eGA_LookAt)) player.Say("He looks like he is hungry.");
else Unhandled();
}
```

In this example, you get a default reaction for everything but look at. The optional parameter is only used internally to make the function work with the door scripts.

Translation

To make the verbs work with translations, strings are being used to define the button graphics, hotkeys and so on. If you like to customize your game or get it translated, you need to take a closer look at the function AdjustLanguage.
The functions getInteger and clearToSpace are being used to extract the informations from those white space seperated strings.

int getInteger ();

String clearToSpace(String text);

function TranslateAction(int action);

function AdjustLanguage();

getInteger

```
int getInteger();
```

This function returns the first integer value of the internal variable "numbers" and calls the function clearToSpace to overwrite that number with blank characters.

See also: clearToSpace

clearToSpace

```
String clearToSpace(String text);
```

This function overwrites the first value or word from the given string with blank characters. Afterwards the new string is being returned.

See also: getInteger

TranslateAction

```
function TranslateAction(int action, int tr_lang);
```

This function defines the text for the verb buttons, e.g. if you click on the talk verb button, "Talk to" is being displayed in the action/status bar. The second parameter defines the returned language. If you want to customize this text, you have to edit this function.

AdjustLanguage

```
function AdjustLanguage();
```

This function has to be called from inside the global script's game_start() function. It sets up everything related to the verb buttons, so you need to take a look at this, if you want to customize your GUI. It is also import to understand, how this function works, if you want to get you game translated.

If you take a closer look at this function, you will notice the following lines:

```
SetActionButtons(eGA_GiveTo, "a_button_give    0  1  2 Gg");
SetActionButtons(eGA_PickUp, "a_button_pick_up 1  7  8 Pp");
SetActionButtons(eGA_Use,    "a_button_use     2 13 14 Uu");
```

```
[...] and so on.
```

Your verb buttons are initialized here, by calling the function SetActionButtons. The first parameter is the action being called by the button, the second one is a string containing the following informations: Name, GUI-button ID, Spriteslot normal, Sprite slot highlighted, Keyboard-Shortcut.

So all those parameters are encapsulated inside a string, sepereated by white spaces. It doesn't matter, how many spaces are used to seperate the parameters, as long as it is at least a single blank character.

This line

```
SetActionButtons(eGA_GiveTo, "a_button_give    0  1  2 Gg");
```

tells the AGS:

The button for the action eGA_GiveTo is named "a_button_give" (also this information is not being used outside of this function).

The buttons has the GUI-ID 0. If you take a look at the GUI "gMaingui", you can see several buttons. The one with the ID 0 will be used for the action you define here.

The button will use the spriteslot 1 as the default graphic and spriteslot 2, if it's highlighted. This can be a little bit confusing, since if you look at gMaingui, those graphics have already been assigned. But you also need to define the graphics slots in this function, because eventually these are the ones being used.

The last parameter defines the hotkey for this action.

You might wonder, why this function overrides the values of gMaingui. But in some other languages the translation for use could be a very long word, so you might want to swap it with something else. E.g. in german "use" means "Benutze", so you need more space for the verb. But "pick up" can be translated to "nimm", so you save some space here. Now in your translation file (e.g: german.tra), you can simply swap the button with GUI of the english "pick up" with the german "Benutze".

```
a_button_use     1 794 795 Bb
```

This function also sets up the fonts for every written text (Option GUI, Status Bar and so on). This is done by this string:

```
font_info=GetTranslation("font_320: 1  0  0  0  0  0  0  3  0  0  0  0  0  3
3");
```

Each numer stands for a text element. The first number is the spoken text, the second number defines the font in the status bar. The other columns are explained in the code itself, if you'd like to customize them.

See also: SetActionButtons

Extensions

function RemoveExtension();

function AddExtension(char extension);

char Extension();

char ExtensionEx(int index, String name);

function OpenCloseExtension(int door_id);

function VariableExtensions();

Extensions are used to define the default action for the right-click. You can add extensions to characters, hotspots, objects and inventory items. To add an extension, e.g. chose an object in the room editor and take a look at the description (not the name). In the sample room, we have an object, called Cup. In addition to the name we have an angle bracket and the letter p:

```
Cup>p
```

The bracket acts as a seperator for the extension, the letter tells the script, which default action to use. By default, the

template knows about the following extensions:

```
n: nothing / default
g: give to
p: pick up
u: use
o: open
l: look at
s: push
c: close
t: talk to
y: pull
v: variable extension
e: exit
```

If you like to customize or add these extensions, take a look at the function CheckDefaultAction.

You don't have to add an extension for every object and hotspot. The template also adds some default actions on it's own. The default action for Characters is "talk to", for Hotspots and Objects, it's look at.
Inventory items are handled a little differently, the right-click always causes "look at", no matter what. If you left-click an item, it's usually "use with". But if you have added the extension "u", the action will be simply "use".
Clicking the verb button "use" and the item afterwards would cause the same action. But it could seem a little bit unpredictable, whether an item can used by a verb button or not. With this shortcut you can make things a little bit easier.
You can see this behaviour in the sample room, when opening the letter. Otherwise you would have needed something else to interact with it. But with the use-extension, it is getting opened by a single left-click. The exit extension is covered in the following chapter.

See also: CheckDefaultAction

RemoveExtension

```
function RemoveExtension();
```

Used to remove the extension from a location (Hotspots, Objects etc.), so it doesn't get displayed in the status bar.

AddExtension

```
function AddExtension(char extension);
```

Used to add a default extension in case the location doesn't have one.

Extension

```
char Extension();
```

Returns the first extention of a location.

ExtensionEx

```
char ExtensionEx(int index, String name);
```

Returns the n-th extension of the given string. This is currently used for exit extensions.

OpenCloseExtension

```
function OpenCloseExtension(int door_id);
```

Used in combination with the door scripts. This function returns a close extension, if the door with the given id is open and vice versa.

VariableExtensions

```
function VariableExtensions();
```

This function is called, if you have have set "v" as an extension for a certain location. Currently it is used for the OpenClose extension, but of course you can add your own variable extensions here, for example "turn on / turn off".