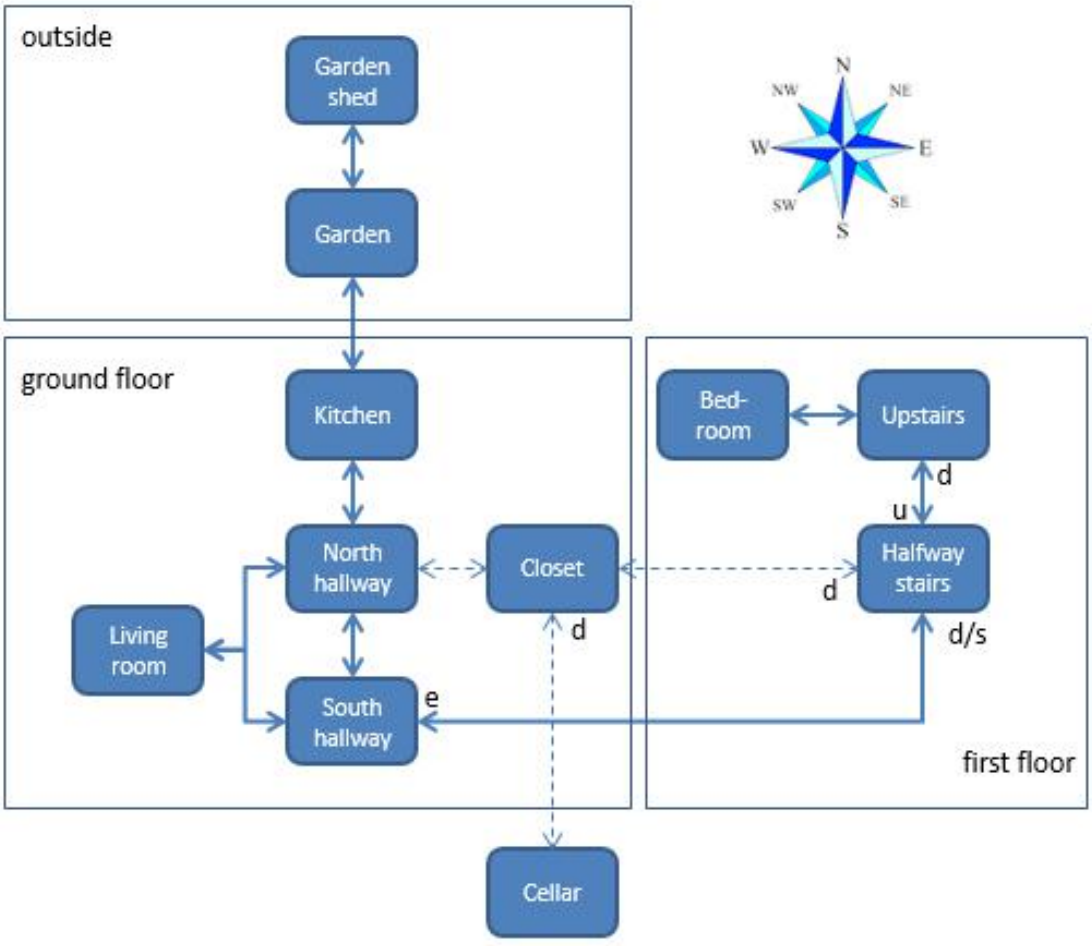
Part 3 – a sample story

Now that we've defined a vocabulary and took care of some basic requirements, we are ready to develop a sample story. Our purpose is not to create an award winning story, but to illustrate how to make locations, objects and timers and how they interact.

We won't make a full size adventure game (I'm not a great author anyway). It will be sort of a first level where you are in a house and must make your way to the cellar. The tutorial game will end when you descend the stairs to the cellar.

The map:



The locations

There are 11 locations. We will enter them in the story file with their long descriptions, short descriptions and directions. Actually, there are 12, we reserve on location, l_storage, to store objects that are removed from play. l_storage is not accessible to the player.

Some locations require non-standard handling of certain directions:

- when going east from the living room, we want to go back to the location we came from, south hallway or north hallway (we use an attribute to remember where we came from);
- going east from north hallway is not possible;

- going north from kitchen to garden is not possible when the kitchen door is locked;
- when the user has not examined the loose step on halfway stairs, going down from halfway stairs will lead to south hallway. After the user examines the loose step, going down will lead to the closet and south will lead to the south hallway.

The locations' code is below. With each location, we will explain new functionality, if any. Some locations also refer to objects, this code will be discussed with the object descriptions.

south hallway

```
$LOCATION l_hallway_south
 DESCRIPTIONS
  d_sys "the south hallway"
  d_longdescr   "You are in the south hallway. To the west is a passage to the /
         living room. To the east are stairs leading up. The hallway /
         continues to the north."
  d_shortdescr  "South hallway"
 EXITS
  n -> l_hallway_north
  w -> l_living_room
  u -> l_halfway
  e -> l_halfway
 TRIGGERS
  "examine [l_hallway_south]"  -> o_player.t_look
  "west"          -> t_west
  "north"         -> t_north
  t_entrance
   move(o_stairs, %this)  # must be able to refer to stairs
   nomatch()
  t_west
   # remember where we came from
   l_living_room.r_back = %this
   nomatch()
  t_north
   # must be able to refer to the closet door
   o_closet_door.r_direction = east
   o_closet_door.r_access    = o_closet_door.d_closet
   move(o_closet_door, l_hallway_north)
   nomatch()
END_LOC
```

Description d_sys is the system description. It is a predefined common description and is used by the parser to map the user input to objects and locations.
To elaborate a bit, the user input is translated from a text string to separate words. The words are looked up in XVAN's word table and replaced by their word id (a number). Next, groups of word ids are held against the location and object tables and mapped on location or object ids. To map the word ids to object/location ids the parser compares them to the word ids from d_sys. As an example, the combination of two word ids for "south" and "hallway" will be mapped to one location id for l_hallway_south. When an object or location has no d_sys description, it cannot be referred to by the user.

Do not forget to include the article in the system description (d_sys). The compiler will strip it and store it separately. Whenever you include [the] or [a] wildcards in a string followed by a location or an object, the interpreter will check whether it has to print an article or not. If you did not include the article in the system description it won't print it. If you did include the article in d_sys but don't use [the] or [a] in a string, the article will not be printed either.
A slash '/' in a string tells the compiler to skip the next <cr> and spaces. It is used for formatting long text strings so they are better readable in the source file.
It may seem a bit unusual to move around the stairs (in trigger t_entrance) but this is just how we model the world. There are several locations from which the stairs are accessible. We could have created individual stair objects in different locations but that would require more code to keep them in sync. The net effect for the person playing the story will be the same and this makes our coding effort easier.
Trigger t_north is used to move around the closet door. It's like moving the stairs but a bit more complicated and will be explained with the closet door object.

north hallway

```
$LOCATION l_hallway_north
 DESCRIPTIONS
  d_sys "the north hallway"
  d_longdescr   "You are in the north hallway. To the west is a passage to the /
          living room. The hallway continues north to the kitchen."
  d_shortdescr  "North hallway"
 EXITS
  n -> l_kitchen
  s -> l_hallway_south
  w -> l_living_room
 TRIGGERS
  "examine [l_hallway_north]"  -> o_player.t_look
  "west"                       -> t_west
  t_west
   # remember where we came from
   l_living_room.r_back = %this
   nomatch()
END_LOC
```

living room

```
$LOCATION l_living_room
 DESCRIPTIONS
  d_sys "the living room"
  d_longdescr   "This is the living room. It's completely abandoned. There is an /
           exit to the hallway to the east."
  d_shortdescr  "Living room"
 EXITS
  # no exits
 ATTRIBUTES
  r_back = l_hallway_south
 TRIGGERS
  "examine [l_living_room]"      -> o_player.t_look
  "[dir]"-> t_go
  t_go
   if equal(%dir, east) then
    move(o_player, r_back)
    entrance(r_back)
    disagree()
   else
    nomatch()
   endif
END_LOC
```

We see that the living room location has a “%dir” trigger, as does the player object. We don’t know the exact order in which objects that are in scope get the user input, but it is ensured that a containing object gets it before its contained objects do. So the location always is the first to get the user input. In our case, because we know the location gets to process the %dir command first, our setup with the r_back attribute will work. In case the user enters any other direction than East, the nomatch() will make the player object’s t_move trigger to further process the user input.

kitchen

```
$LOCATION l_kitchen
 DESCRIPTIONS
  d_sys "the kitchen"
  d_longdescr   "This is the kitchen. There is not much here. The /
          hallway is to the south."
  d_shortdescr  "Kitchen"
 EXITS
  s -> l_hallway_north
 TRIGGERS
  "examine [l_kitchen]" -> o_player.t_look
  "n"     -> t_north
  “s”     -> t_south
 t_entrance
   printcr(d_shortdescr)
   if not(testflag(f_seenbefore) AND not(testflag(o_player.f_verbose))) then
    printcr(d_longdescr)
   endif
   move(o_kitchen_door, %this)  # must be able to refer to the door
  t_north
   if testflag(o_kitchen_door.f_locked) then
    printcr("The kitchen door is locked.")
   else
    if not(testflag(o_kitchen_door.f_open)) then
     printcr("[[opening the kitchen door first]")
     setflag(o_kitchen_door.f_open)
     newexit(l_kitchen, north, l_garden)
    endif
    move(o_player, n)    # also updates current location
    entrance(l_location)
   endif
   disagree()  # prevent o_player.t_move to execute the "n" command
  t_south
   # must be able to refer to the closet door
   o_closet_door.r_direction = east
   o_closet_door.r_access    = o_closet_door.d_closet
   move(o_closet_door, l_hallway_north)
   nomatch()
END_LOC
```

The valdir() function checks for a valid direction (exit from the current location).

When the user wants to go north, the t_north trigger is fired. If the door is locked, we print a rejection message. If it is unlocked but closed, we don't print a "the door is closed" rejection message, but open the door for the player. Note the last disagree(). It tells the interpreter to stop and not offer the user's command to other objects. If we forget it, the command will also be sent to the o_player object who will execute it. Since at that moment we already are in the garden (t_north has already moved the player object to the north), the player will finally end up in the shed.

Trigger t_south is used to move around the closet door. It’s like moving the stairs but a bit more complicated and will be explained with the closet door object..

closet

```
$LOCATION l_closet
 DESCRIPTIONS
  d_sys "the closet"
  d_longdescr   "You are in a dark closet below the staircase. To the west is /
          the closet door, which is closed."
  d_shortdescr  "Closet"
 EXITS
  u -> l_halfway
  d -> l_cellar
 TRIGGERS
  "examine [l_closet]"   -> o_player.t_look
  "down"          -> t_down
  t_entrance
    printcr(d_shortdescr)
    printcr(d_longdescr)
    if not(testflag(o_trapdoor.f_hidden)) then
      printcr("Visible exits are up and down.")
    else
      printcr("The only visible exit is up.")
    endif
  t_down
    if testflag(o_trapdoor.f_hidden) then
      printcr("The carpet is blocking your way down.")
      disagree()
    else
      if not(testflag(o_trapdoor.f_open)) then
        printcr("The trapdoor is closed.")
        disagree()
      else
        nomatch()  # let o_player.t_move handle this
END_LOC
```

Flag f_hidden is a predefined common flag. When set, the object or location is treated by the parser as not visible, so the player won't be able to refer to it.

cellar

```
$LOCATION l_cellar
 DESCRIPTIONS
  d_sys " the cellar"
  d_burning      "You walk down the stairs into the cellar. Down below you see /
            the red glow of a fire. As you walk down further, it gets hotter /
            and hotter. You realize you will be fried if you continue and you /
            hurry back up the stairs."
  d_not_burning          "There is still a lot of smoke in the cellar, but through the /
            hazes you can make out an old workbench to the east and a door /
            to the north."
  d_shortdescr  "Cellar"
  d_end          "
/          ***** this is the end of the tutorial *****
/          “
EXITS
```

```
  up -> l_closet
 FLAGS
  f_tried_before = 0
 TRIGGERS
  "examine [l_cellar]" -> o_player.t_look
  t_entrance
    if testflag(o_flames.f_extinguished) then
      printcr(d_not_burning)
      printcr(d_end)
      quit()
    else
      if not(testflag(f_tried_before)) then
        setflag(f_tried_before)
        printcr(d_burning)
      else
        printcr("There's flames down there, remember?")
      endif
      move(o_player, u)
    endif
    agree()
END_LOC
```

halfway stairs

```
$LOCATION l_halfway
 DESCRIPTIONS
  d_sys "halfway"
  d_longdescr   "You are now halfway up the stairs. The stairs continue up /
          to the north and down to the south."
  d_shortdescr  "Halfway stairs"
  d_up_closed   "When you walk further up the stairs one of the steps makes /
          a hollow sound. You try to pinpoint it but get no further /
          than that it is somewhere in the upper half of the stairs.
/          "
  d_up_open     "You carefully step over step 11, so you don't fall down /
          into the closet.
/          "
 EXITS
  n -> l_upstairs
  u -> l_upstairs
  s -> l_hallway_south
  d -> l_hallway_south
 TRIGGERS
  "examine [l_halfway]"           -> o_stairs.t_exa
  "up"   -> t_up
  "north"          -> t_up
  “down”          -> t_down
  t_entrance
    print(d_longdescr)
    move(o_stairs, %this)  # must be able to refer to stairs
    agree()
  t_up
     if testflag(o_button.f_pressed) then
```

```
    # step 11 is open
    printcr(d_up_open)
  else
    printcr(d_up_closed)
 t_down
  # must be able to refer to the closet door
  o_closet_door.r_direction = west
  o_closet_door.r_access    = o_closet_door.d_hallway
  move(o_closet_door, l_closet)
  nomatch()
END_LOC
```

Location halfway stairs has its own local t_entrance trigger, because we must move the stairs object to this location when the player enters. If we don't do this, then the user won't be able to refer to the stairs.

Trigger t_down is used to move around the closet door. It's like moving the stairs but a bit more complicated and will be explained with the closet door object..

upstairs

```
$LOCATION l_upstairs
 DESCRIPTIONS
  d_sys "upstairs"
  d_longdescr   "You are upstairs. Behind you, the stairs lead down. There /
          is an exit to the west."
  d_shortdescr  "Upstairs"
  d_down_closed         "When you walk down, one of the steps makes a hollow sound. /
          You try to pinpoint it but get no further than that it is /
          at the top half of the stairs.\n"
  d_down_open           "You carefully step over step 11, so you don't fall down /
           into the closet.\n"
 EXITS
  s -> l_halfway
  d -> l_halfway
  w -> l_bedroom
 TRIGGERS
  "examine [l_upstairs]"          -> o_player.t_look
  "down"          -> t_down
  "south"         -> t_down
  t_entrance
   move(o_stairs, %this) # must be able to refer to stairs
   nomatch()
  t_down
   if testflag(o_button.f_pressed) then
    # step 11 is open
    printcr(d_down_open)
   else
    printcr(d_down_closed)
END_LOC
```

Again, we move the stairs object in a local t_entrance trigger because the player must be able to refer to the stairs. The local t_entrance trigger returns nomatch(), so the common t_entrance trigger will be executed as well.

bedroom

```
$LOCATION l_bedroom
 DESCRIPTIONS
  d_sys "the bedroom"
  d_longdescr    "This location used to be a bedroom a long time ago. But /
             now, there is nothing there. All furniture has been /
             removed."
  d_shortdescr  "Bedroom"
  d_exa"Mounted to the west wall are a water tap and a sink."
 EXITS
  e -> l_upstairs
 TRIGGERS
  "examine [l_bedroom]" -> o_player.t_look
  t_entrance
    printcr(d_shortdescr)
    if not(testflag(f_seenbefore)) or testflag(o_player.f_verbose) then
      # first visit or verbose mode
      setflag(f_seenbefore)
      printcr(d_longdescr)
    endif
END_LOC
```

garden

```
$LOCATION l_garden
 DESCRIPTIONS
  d_sys "the garden", "the hedges", "the hedge"
  d_longdescr   "You are in the garden at the back of the house. East and west /
            there are hedges. To the north is a garden shed."
  d_shortdescr  "garden"
 EXITS
  s -> l_kitchen
  n -> l_shed
 TRIGGERS
  "examine [l_garden]" -> o_player.t_look
 t_entrance
   printcr(d_shortdescr)
   if not(testflag(f_seenbefore) AND not(testflag(o_player.f_verbose))) then
     printcr(d_longdescr)
   endif
   move(o_kitchen_door, %this)  # must be able to refer to the door
END_LOC
```

The garden is also described as the hedge and hedges. When we use [l_garden] in a string, the interpreter will always print it as “garden”, even if the user referred to it as hedge. If we want the interpreter to print the system description that the player used last, then we must set the predefined flag f_swap. Printing l_garden.d_sys will always print the first system description, regardless of f_swap.

shed

```
$LOCATION l_shed
 DESCRIPTIONS
  d_sys "the garden shed"
  d_longdescr   "You are now in the garden shed. The shed hasn't been cleaned /
          for a long time. Maybe never. On the walls you see the nails /
          that were used to hang the garden utensils to. Almost all of /
          them are gone now."
  d_shortdescr  "Garden shed"
 EXITS
  s -> l_garden
 TRIGGERS
  "examine [l_shed]" -> o_player.t_look
END_LOC
```

Why is the shed a location and not an object in the garden? That's just a design choice, it could have been an object as well. Making it an object in the garden is a bit more work though, because all user input will then be offered to the garden as well and we may have to write extra code for t_entrance and to move around (e.g. when in the shed object, "s" will take us to the kitchen).

The objects

Now that we've got the map, let's take a look at the objects.

We have the following objects:

- player
- nst (no such thing)
- kitchen door
- kitchen window
- toaster
- key hole
- rusty key
- glass fragment
- hacksaw
- stairs
- steps
- button (on stairs)
- closet door
- floor (in closet)
- carpet
- trapdoor
- drain pipe in bedroom
- drain pipe in closet
- flames
- water tap
- water
- sink

Before going into the object descriptions, we'll briefly describe the plot of this tutorial game:

- go to the kitchen
- get the toaster and throw it through the window
- look through the kitchen door and notice the key on the outside;
- open the kitchen door
- go inside the shed and get the hacksaw
- go back into the kitchen and get the window fragment
- go to the stairs and find the button near step 11

- move the step and go down into the closet
- cut the carpet with the fragment
- open the trapdoor and see the flames
- cut the drain pipe with the hacksaw
- go up to the bedroom and open the water tap
- go back into the closet and see that the flames are extinguished by the water
- enter the cellar
- end of first level

In the next sections we'll describe the objects, list the code and clarify where necessary.

player

We already addressed the player object in section 2 of the tutorial.

nst

o_nst is the 'no-such-thing' object. It's predefined by the compiler and must be in de story file. It is used with disambiguation rules as explained in part 5 of this tutorial. We'll leave it for now.

it

We want the player to be able to refer to a previous object by "it". We won't use it in this tutorial, but the o_it object is predefined by the compiler and must be in the story file. Don't worry about it.

Note: the o_nst and o_it objects are predefined in the XVAN Starter Kit (o_it as of version 1.1). If you use the Starter Kit, nst and it are taken care of automatically.

kitchen door

Is in the kitchen and leads to the garden. The door is locked and the key is in the key hole on the other side of the door. In the door is a window. The window and key hole are also defined as objects with their own t_entrance triggers.

It was a design decision to not mention the window in the door descriptions, because the window must be broken at some point which would result in outdated descriptions. It is better to let the window object handle this by itself.

```
$OBJECT o_kitchen_door
 DESCRIPTIONS
  d_sys "the kitchen door"
  d_longdescr   "The door is made of wood; it gives access to the garden."
  d_longdescr1 "The door is made of wood; it leads back into the kitchen."
  d_shortdescr  "To the north is a door that leads to the garden."
  d_shortdescr1           "To the south is a door that leads to the kitchen."
  d_no_window           "In the upper half of the door is an opening where /
            a window used to be "
 CONTAINED in l_kitchen
 FLAGS
  f_openable    = 1
  f_lockable    = 1
  f_locked      = 1
 TRIGGERS
  "examine [o_kitchen_door]"   -> t_exa
  "look through [o_kitchen_door]"       -> o_kitchen_window.t_look_through
  "unlock [o_kitchen_door] with [o_rusty_key]"-> t_unlock
  "turn [o_rusty_key]"  -> t_unlock
  "open [o_kitchen_door]"       -> t_open
  "close [o_kitchen_door]"      -> t_close
  t_entrance
    if owns(l_kitchen, %this) then
      printcr(d_shortdescr)
    else
      printcr(d_shortdescr1)
  t_exa
    if owns(l_kitchen, %this) then
      print(d_longdescr)
    else
      print(d_longdescr1)
    endif
    if testflag(f_open) then
      print(" The door is open. ")
    else print(" The door is closed. ")
    endif
    # print info about window and keyhole
    if testflag(o_kitchen_window.f_broken) then
      print(d_no_window)
    else
      print(o_kitchen_window.d_shortdescr)
    endif
    printcr(o_keyhole.d_shortdescr)
    contents(o_keyhole)
  t_unlock
    if not(owns(o_player, o_rusty_key)) and not(owns(o_keyhole, o_rusty_key)) then
      printcr("[[picking up the rusty key first]")
      move(o_rusty_key, o_player)
    endif
    # verb prologue will check if already unlocked
```

```
    if not(owns(o_keyhole, o_rusty_key)) then
      printcr("[[putting the rusty key in the keyhole]")
    endif
    printcr("Ok, the kitchen door is now unlocked.")
    clearflag(f_locked)
  t_open
    # test for already open is done by verb prologue
    if not(testflag(f_locked)) then
      printcr("Ok, the kitchen door is now open")
      setflag(f_open)
      newexit(l_kitchen, north, l_garden)
    else
      printcr("The door seems to be locked.")
  t_close
    # test for already closed is done by verb prologue
    printcr("Ok, the kitchen door is now closed.")
    clearflag(f_open)
    blockexit(l_kitchen, n)
END_OBJ
```

To create and delete exits we use functions newexit() and blockexit().
For this object, we also need verbs "unlock" and "open". And while we're at it, we will create "lock" and " close" as well.
With these verbs we test as many general things (already open/closed/locked/unlocked) in the verb prologue, so we don't have to repeat the same tests in the objects. The general tests do require some additional common flags: f_openable, f_open, f_lockable, f_locked.
verb unlock

```
$VERB unlock
  PROLOGUE
    if not(equal(o_subject, %none)) then
      if not(testflag(o_subject.f_lockable)) then
        printcr("[the] [o_subject] is not something that can be unlocked.")
        disagree()
      else
        if not(testflag(o_subject.f_locked)) then
          printcr("But [the] [o_subject] [o_subject.r_be] not locked.")
          disagree()
        endif
      endif
    endif    # endifs at the end of code  may be omitted
  "unlock"
    printcr("What do you want to unlock?")
    getsubject()
  "unlock [o_subject]"
    printcr("How do you want to unlock [the] [o_subject]?")
    getspec()
  "unlock [o_subject] with [o_spec]"
    printcr("[the] [o_actor] cannot unlock [the] [o_subject] with [the] [o_spec].")
  DEFAULT
    printcr("I only understood you as far as wanting to unlock something.")
ENDVERB
```

verb open

```
$VERB open
  PROLOGUE
    if not(equal(o_subject, %none)) then
      if not(testflag(o_subject.f_openable)) then
        printcr("[the] [o_subject] is not something that can be opened.")
        disagree()
      else
        if testflag(o_subject.f_open) then
          printcr("But [the] [o_subject] [o_subject.r_be] is already open.")
          disagree()
      endif
    endif
  endif
  "open"
    printcr("What do you want to open?")
    getsubject()
  "open [o_subject]"
    printcr("[the] [o_actor] can't open that.")
ENDVERB
```

verb lock

```
$VERB lock
  PROLOGUE
    if not(equal(o_subject, %none)) then
      if not(testflag(o_subject.f_lockable)) then
        printcr("[the] [o_subject] is not something that can be locked.")
        disagree()
      else
        if testflag(o_subject.f_locked) then
          printcr("But [the] [o_subject] [o_subject.r_be] is already locked.")
          disagree()
        endif
      endif
    endif
  "lock"
    printcr("What do you want to lock?")
    getsubject()
  "lock [o_subject]"
    printcr("How do you want to lock [the] [o_subject]?")
    getspec()
  "lock [o_subject] with [o_spec]"
    printcr("[the] [o_actor] cannot lock [the] [o_subject] with [the] [o_spec].")
  DEFAULT
    printcr("I only understood you as far as wanting to lock something.")
ENDVERB
```

verb close

```
$VERB close
  PROLOGUE
    if not(equal(o_subject, %none)) then
      if not(testflag(o_subject.f_openable)) then
        printcr("[the] [o_subject] is not something that can be closed.")
        disagree()
      else
        if not(testflag(o_subject.f_open)) then
          printcr("But [the] [o_subject] [o_subject.r_be] is already closed.")
          disagree()
      endif
    endif
  endif
  EPILOGUE
    if not(islit(o_player)) then
      # they may have closed a container with the light source
      printcr("It is now pitch black.")
      disagree()
  "close"
    printcr("What do you want to close?")
    getsubject()
  "close [o_subject]"
    printcr("[the] [o_actor] can't close that.")
ENDVERB
```

You notice that for verb close we also have an epilogue. In the epilogue we check if closing the subject made the light source invisible. For example, if the player puts his flashlight in a box and closes it, it will become dark. The epilogue will detect this.

kitchen window

As already mentioned with the kitchen door object, the kitchen window is an autonomous object, because it will have different behavior once it is broken. It makes less sense to code this all with the door object.

```
$OBJECT o_kitchen_window
 DESCRIPTIONS
  d_sys "the kitchen window"
  d_longdescr   "The window is made of glass, which somehow doesn't /
           surprise you."
  d_shortdescr  "In the upper half of the door is a window "
  d_smash_no   "You smash the window with your fist, but with no /
          success. You need something heavy to break the /
          window."
  d_broken       "\nScattered over the floor is a broken window that /
           once was a part of the door."
  d_look_glass  "Through the window you see the garden. At the far /
           face to the window and try to look down but can't /
           see right behind the door. If you could only stick /
           your face further through."
  d_look_no_glass        "Because the window is no longer there, you can stick /
           your head through the hole. There's a rusty key in /
           the outside of the keyhole!"
  d_climb         "You don't fit through the window. It's way too small (or /
           you are too big)."
 CONTAINED in o_kitchen_door
 FLAGS
  f_broken = 0
 TRIGGERS
  "examine [o_kitchen_window]"         -> t_exa
  "look through [o_kitchen_window]"   -> t_look_through
  "break [o_kitchen_window]"  -> t_break_no
  "break [o_kitchen_window] with [o_spec]"     -> t_break
  "throw [o_subject] [prepos] [o_kitchen_window]"      -> t_throw
  "climb through [o_kitchen_window]" -> t_climb
  "go through [o_kitchen_window]"      -> t_climb
  t_entrance
    if testflag(f_broken) then
      printcr(d_broken)
  t_look_through
    if testflag(f_broken) then
      printcr(d_look_no_glass)
      clearflag(o_rusty_key.f_hidden)
    else
      printcr(d_look_glass)
  t_break_no
    printcr(d_smash_no)
```

```
  # break and throw cannot be the same trigger because they
  # have their subject and specifier reversed.
  t_break
   print("You throw [the] [o_spec] at the window")
   if not(testflag(o_spec.f_heavy)) then
    printcr(", but it bounces back. It obviously isn't /
          heavy enough.")
    move(o_spec, l_kitchen)
   else
    printcr(" and it goes straight through. The window /
          is shattered all over the floor. One of the /
          glass fragments is a bit larger than the rest.")
    move(o_spec, l_garden)
    setflag(f_broken)
    move(%this, l_kitchen)
    move(o_fragment, l_kitchen)
   endif
  t_throw
   # may only work for 'at' and 'through'
   if equal(%prepos, at) OR equal(%prepos, through) then
    print("You throw [the] [o_subject] at the window")
    if not(testflag(o_subject.f_heavy)) then
     printcr(", but it bounces back. It obviously isn't /
           heavy enough.")
     move(o_subject, l_kitchen)
    else
     printcr(" and it goes straight through. The window /
           is shattered all over the floor. One of the /
           glass fragments is a bit larger than the rest.")
     move(o_subject, l_garden)
     setflag(f_broken)
     move(%this, l_kitchen)
     move(o_fragment, l_kitchen)
    endif
   else
    nomatch()
  t_climb
   printcr(d_climb)
   disagree()
END_OBJ
```

Triggers t_break and t_throw are almost identical. But because they have subject and specifier reversed, we must code separate triggers. We also need an additional common flag for these triggers: f_heavy.

We must also define some additional verbs in our vocabulary: break, throw and climb.

verb climb

```
$VERB climb
  "climb"
   printcr("What do you want to climb?")
   getsubject()
```

```
  "climb [o_subject]"
    printcr("[the] [o_subject] is not something to climb.")
  "climb [prepos] [o_subject]"
    printcr("[the] [o_actor] cannot climb [prepos] [the] [o_subject].")
  DEFAULT
    printcr("I only understood you as far as willing to climb something.")
ENDVERB
```

verb break

```
VERB break SYNONYM destroy
 "break"
  printcr("What do you want to break?")
  getsubject()
 "break [o_subject]"
  printcr("[the] [o_actor] can't break [the] [o_subject].")
 "break [o_subject] with [o_spec]"
  printcr("[the] [o_actor] can't break [the] [o_subject] with [the] [o_spec].")
 DEFAULT
  printcr("I only understood you as far as wanting to break something.")
ENDVERB
```

verb throw

```
VERB throw
 PROLOGUE
  # actor must hold the subject
  if not(equal(o_subject, %none)) then
   if not(owns(o_actor, o_subject)) then
    printcr("[the] [o_actor] must be holding [the] [o_subject] first.")
    disagree()
   endif
  endif
 "throw [o_subject] [dir]",
 "throw [o_subject] to [dir]"
  if valdir(l_location, %dir) then
   move(o_subject, %dir)
   printcr("Thrown.")
  else
   printcr("[the] [o_subject] bumps to the [dir] wall and falls on the floor.")
   move(o_subject, l_location)
 "throw [o_subject] [prepos] [o_spec]"
  printcr("Throwing [the] [o_subject] [prepos] [the] [o_spec] won't work.")

 DEFAULT
  printcr("I only understood you as far as wanting to throw something.")
ENDVERB
```

We also coded some standard functionality in the throw verb for throwing objects in a particular direction.

The strings "throw [o_subject] to [dir]" immediately follows "throw [o_subject] [dir]". This means that the code that follows applies to both commands.

Next, we'll continue with the keyhole object.

keyhole

```
$OBJECT o_keyhole
 DESCRIPTIONS
  d_sys "the keyhole"
  d_longdescr   ""
```

```
  d_shortdescr  "and you also see a keyhole."
  d_peek        "You peek through the keyhole but you cannot see a thing. /
          something on the other side of the keyhole blocks your view."
  d_look        "You see the garden. At the far east end, there is a /
          garden shed."
 CONTAINED in o_kitchen_door
 TRIGGERS
  "examine [o_keyhole]"          -> t_look_through
  "look through [o_keyhole]"     -> t_look_through
  t_entrance
    # don't print there is a keyhole when entering the room
    agree()
  t_look_through
    if owns(o_keyhole, o_rusty_key) then
      if cansee(o_player, o_rusty_key) then
        printcr("You can't, since there is a key in the keyhole.")
      else
        printcr(d_peek)
      endif
    else
      printcr(d_look)
    endif
    disagree()
END_OBJ
```

With the keyhole object we don't use d_longdescr and d_shortdescr. For examining we use t_look_through and default t_entrance will always use d_sys because the keyhole is a part of the door that cannot be removed ("…. [this]…." Will print d_sys from the current object).

This part of the common t_entrance trigger applies for the keyhole object:

```
    else
     if not(owns(o_player, %this, 0))  then
      # it´s not (in) some object the player carries (0 means all levels of containment)
      setflag(f_seenbefore)
      print("There is [a] [this] [r_preposition] [the] ")
      print(owner(%this))
      printcr(".")
     endif
```

rusty key

```
$OBJECT o_rusty_key
 DESCRIPTIONS
  d_sys "the rusty key"
  d_longdescr   "An old rusty metal key."
  d_shortdescr  "An old rusty metal key."
 CONTAINED in o_keyhole
 FLAGS
  f_takeable = 1
  f_hidden   = 1   # key is in the other side of the keyhole
 TRIGGERS
  "inventory"    -> t_i
  "examine [o_rusty_key]"        -> t_exa
END_OBJ
```

glass fragment

```
$OBJECT o_fragment
 DESCRIPTIONS
  d_sys "the glass fragment", "the shard", "the splinter"
  d_longdescr   "The fragment is about 5 inches long and has a sharp edge."
  d_shortdescr  "There is a glass fragment here."
  d_carpet      "You cut the carpet along its sides and it comes loose /
            from the floor, revealing a trapdoor!"
 CONTAINED in l_storage
 FLAGS
  f_takeable = 1
 TRIGGERS
  "inventory"    -> t_i
  "examine [o_fragment]"        -> t_exa
  "cut [o_carpet] with [o_fragment]"    -> t_cut
  t_cut
    if not(owns(o_player, %this)) then
      printcr("[[picking up the fragment first]")
    endif
    if not(testflag(o_carpet.f_cut)) then
      setflag(o_carpet.f_cut)
      move(o_trapdoor, l_cellar)
      move(o_carpet, o_player)
      setflag(o_carpet.f_bypass)
      printcr(d_carpet)
    else
      printcr("You already did that.")
END_OBJ
```

We don't want the carpet lying around after cutting it, so we make the player pick it up in the cut action.

For the glass fragment we need to define the verb "cut".

verb cut

```
$VERB cut SYNONYM saw
  "cut"
    printcr("What do you want to cut?")
    getsubject()
  "cut [o_subject]"
    printcr("How do you want to cut [the] [o_subject]?")
    getspec()
  "cut [o_subject] with [o_spec]"
    printcr("[the] [o_actor] cannot cut [the] [o_subject] with [the] [o_spec].")
  DEFAULT
    printcr("I only understood you as far as wanting to cut something.")
ENDVERB
```

toaster

The toaster object is in the kitchen. We need the toaster to break the window in the kitchen door so we can reach the key that is on the outside of the door

The toaster object

```
$OBJECT o_toaster
 DESCRIPTIONS
   d_sys "the toaster"
   d_longdescr    "An old toaster, quite heavy. The power cord /
              has been cut off."
   d_shortdescr  "There's a toaster here."
 CONTAINED in l_kitchen
 FLAGS
   f_takeable      = 1
   f_heavy         = 1
 TRIGGERS
   "inventory"    -> t_i
   "examine [o_toaster]"            -> t_exa
END_OBJ
```

We set flag f_heavy for the toaster so it can be used to break the window in the kitchen door.

hacksaw

```
$OBJECT o_hacksaw
 DESCRIPTIONS
  d_sys "the hacksaw", "the saw"
  d_longdescr   "This is just an ordinary hacksaw. It can be used /
            to saw metal objects. The saw looks a bit worn, /
            but it probably will last for one more saw job."
  d_shortdescr  "There is a hacksaw here."
  d_no_saw      "The saw is pretty worn. It will probably last for /
            one more saw job and your planned action is unlikely /
            to be that job."
  d_worn        "The saw is completely worn out. Whatever you are going to /
            do with it, it won't be a saw job."
 CONTAINED in l_shed
 FLAGS
  f_takeable     = 1
  f_worn         = 0
 TRIGGERS
  "inventory"    -> t_i
  "examine [o_hacksaw]"         -> t_exa
  "saw [o_subject] with [o_hacksaw]"    -> t_saw
  t_exa
    if testflag(f_worn) then
      printcr(d_worn)
    else
      printcr(d_longdescr)
    endif
    disagree()
  t_saw
    if not(equal(o_subject, o_drain_pipe_closet)) then
      if testflag(f_worn) then
        printcr(d_worn)
      else
        printcr(d_no_saw)
END_OBJ
```

We only allow the user to use the hacksaw once, to cut the drain pipe in the closet. For all other situations we have defined rejection messages.

We do not make a separate “saw” verb but define a synonym for the “cut” verb instead.

stairs

The stairs is an object that will be available in the following locations:

- l_hallway_north
- l_halfway;
- l_upstairs.

From within these locations, the player must be able to refer to the stairs. The stairs will be moved to the location once the player enters it.

The first time the examine command is given, it will only work if the player is in location l_halfway (halfway up the stairs). Once examined from location l_halfway, the examine command will also work from the other two locations (we use flag f_exa to check for this).

```
$OBJECT o_stairs
 DESCRIPTIONS
  d_sys "the stairs", "the staircase"
  d_exa"You see nothing special about the stairs."
  d_exa_hollow"It looks just like a staircase with one /
            step that sounds hollow when stepped on."
  d_longdescr   "It's a wooden staircase. There are 15 steps. You can refer /
            to a particular step with 'step <number>'."
  d_shortdescr  ""  # included in room description for hallway south and halfway
  d_count         "There are 15 steps. You can refer to a particular step with /
            'step <number>'."
  d_cant_see    "It's hard to get a good view from here. If you were halfway /
            the stairs you would have a better view."
 CONTAINED in l_hallway_south
 FLAGS
  f_exa = 0  # Not yet examined.
 TRIGGERS
  "examine [o_stairs]"   -> t_exa
  "look at [o_stairs]"     -> t_exa
  "count [o_steps]"       -> t_count
  t_entrance
    agree()  # Must execute t_entrance for contained objects (steps).
  t_exa
    if (equal(l_location, l_hallway_south) OR equal(l_location, l_upstairs))
       AND not(testflag(f_exa)) then
      printcr(d_cant_see)
    else
      # we are halfway
      # if they have not yet heard the hollow sound, we don't mention it
      if testflag(l_upstairs.f_seenbefore) then
        setflag(f_exa)
        printcr(d_exa_hollow)
      else
        printcr(d_longdescr)
  t_count
    printcr(d_count)
END_OBJ
```

Now, we also need a verb 'count':

```
$VERB count
  "count"
    printcr("1 2 3")
  "count [o_subject]"
    printcr("[the] [o_subject] is not something that can be counted.")
  DEFAULT
    printcr("I only understood you as far as wanting to count something.")
ENDVERB
```

steps

The steps object is part of the stairs. There are 15 steps and they can be referred to individually (but there is only one steps object). Referring to steps goes by “step <number>”. The number entered by the player is captured in the %ord wildcard, where ord stands for ordinal.

A little something about number wildcards
XVAN has two number wildcards: %value and %ord. The difference is best explained with some examples:
%ord captures ordinal numbers, something with a certain order. “examine step 5” will cause the number 5 to be stored in %ord.
%value captures values, all other numbers. “set dial to 1234” or “enter 1234 on keypad” will store the number 1234 in %value.
Step 11 is a special step, as soon as the player examines it, he will be notified that there is a button next to the step.
the steps object

```
$OBJECT o_steps
 DESCRIPTIONS
  d_sys "the steps", "the step"
  d_longdescr   "There's a tiny button on the side of the step."
  d_shortdescr  ""
  d_15  "There are only 15 steps."
  d_which       "If you want to do something to a specific step, please refer to /
          the step as 'step <number>'."
 d_moved_11   "Step 11 has disappeared, revealing a passage down."
CONTAINED in o_stairs
FLAGS
  f_swap = 1  # always print the d_sys last referred to by the user
TRIGGERS
  "examine [o_steps]"   -> t_exa
  "examine [o_steps] [ord]"      -> t_exa_step
 t_entrance
   agree()
 t_exa
   if not(trigger(o_stairs.t_exa)) then
     disagree()
 t_exa_step
   if (equal(l_location, l_hallway_south) OR equal(l_location, l_upstairs))
       AND not(testflag(o_stairs.f_exa)) then
     printcr(o_stairs.d_cant_see)
     disagree() # stop
   endif
   if lt(%ord, 1) or gt(%ord, 15) then
     printcr("Steps are numbered from 1 to 15.")
   else
     # step 11 gives access to the closet
     if equal(%ord, 11) then
       if not(testflag(o_button.f_pressed)) then
         printcr(d_longdescr)
         clearflag(o_button.f_hidden)
       else
```

```
      printcr(d_moved_11)
    else
     printcr("You see nothing special about step [ord].")
 t_default
  if equal(o_subject, o_steps) then
   printcr(d_which)
   disagree()
  else
   nomatch()  # this is important for default verb code
END_OBJ
```

The trigger() function is used to execute a trigger from another object or location. It returns true or false. When the trigger to be executed returns disagree, the trigger() function will return false.

The t_default trigger is a special system defined trigger. If none of the triggers of an object fired, the t_default trigger - if present - will fire. We use it here to catch all actions on the steps that we did not foresee and print a message on how to refer to the steps. Since the o_steps object receives ALL user input, it must check the subject and only reply if the subject is o_steps. If not, it is very important to return a nomatch() result because otherwise the interpreter will see that atrigger fired and it will not call verb code.

button

The button is hidden until the player examines step 11.

```
$OBJECT o_button
 DESCRIPTIONS
  d_sys  "the button"
  d_longdescr    "A round button in the same color as the stairs. You have to look /
            really close to notice it."
  d_shortdescr   "
/           There's a tiny button on the side of step 11."
  d_press          "As you press the button, step 11 retracts a bit, lowers /
            about an inch and then slides backwards out of sight, /
            revealing a passage down into the closet!"
 CONTAINED in l_halfway
 FLAGS
  f_hidden        = 1
  f_pressed       = 0
 TRIGGERS
  "examine [o_button]" -> t_exa
  "examine [o_stairs]" -> t_exa_stairs
  "press [o_button]"   -> t_press
  t_entrance
   if not(testflag(f_hidden)) then
    # the button is visible
    if not(testflag(f_pressed)) then
     printcr(d_shortdescr)
    else
     printcr(o_steps.d_moved_11)
  t_exa_stairs
   if not(testflag(f_hidden)) then
    printcr(d_shortdescr)
```

```
 t_press
  if testflag(f_pressed) then
   printcr("Nothing happens.")
  else
   printcr(d_press)
   o_player.r_score += 50
   printcr("")
   printcr("[[Your score just went up by 50 points!]")
   setflag(f_pressed)
   blockexit(l_halfway, d)
   newexit(l_halfway, d, l_closet)
  endif
  disagree()
END_OBJ
```

closet door

The closet door cannot be opened. Access to the closet is through the staircase when step 11 is open.

```
$OBJECT o_closet_door
 DESCRIPTIONS
  d_sys "the closet door"
  d_longdescr   "The closet door seems to be locked."

  d_shortdescr  "To the [r_direction] is a door that gives access /
          to [r_access]."
  d_closet      "a closet under the stairs"
  d_hallway     "the north hallway"
  d_no_unlock   "[the] [o_spec] does not fit."
 CONTAINED in l_hallway_north
 ATTRIBUTES
  r_direction   = east
  r_access      = d_closet
 FLAGS
  f_openable    = 1
  f_lockable    = 1
  f_locked      = 1
 TRIGGERS
  "east"-> t_east
  "examine [o_closet_door]"     -> t_exa
  "open [o_closet_door]"        -> t_locked
  "unlock [o_closet_door] with [o_rusty_key]"   ->t_unlock
  t_east
   printcr("The closet door is closed.")
   disagree()
  t_locked
   printcr(d_longdescr)
  t_unlock
   printcr(d_no_unlock)
END_OBJ
```

The closet door is moved around between locations l_hallway_north and l_closet. We see that its shortdescr description contains two attributes: direction and access. Depending on whether the

closet door object is in the north hallway or the closet, we change the value of the attributes. This ensures that in t_entrance the correct description will be printed:
“To the east is a door that gives access to a closet under the stairs.”
Or
“To the west is a door that gives access to the north hallway.”
But, wait a second. I understand you want to move the closet door to the locations where it must be in scope. I compared it to the stairs object that is moved around as well, and the stairs object is moved in the t_entrance trigger from the location where it must end up whereas the closet door object is moved in a special trigger from the location that the player is leaving.
=> when the player is moving from south hallway to halfway stairs, the stairs object is moved to halfway stairs in t_entrance from halfway stairs.
=> when the player is moving from the kitchen to hallway north, the closet door object is moved in t_south from the kitchen and NOT in t_entrance from hallway north.
Why?
There’s a good reason for that. The stairs object has no actions for its t_entrance trigger (other than agree). The closet door’s t_entrance trigger must print a description. Remember that in the player’s t_move trigger the entrance(l_location) function is called? This function creates a list of all objects whose t_entrance must be called. If one of these t_entrance triggers adds another object (like moving the stairs or the closet door) this object will not be on the list and its t_entrance trigger will not be called. For the stairs this is not an issue, because its t_entrance doesn’t do anything, but for the closet door it is. We solved it by moving the closet door from the current location if the player is going to a location from where he must be able to refer to the closet door.
But, the living room also leads to the north hallway does not have a trigger to move the closet door to the north hallway? Right, but the only way you can go from the living room to the north hallway is when you came from the north hallway first. So the closet door will already be there.

floor

The floor is sort of a scenery object. We want the user to be able to refer to the floor, but is has all the default replies. We override the common t_entrance trigger with a local one that doesn’t do anything, because we don’t want the floor to be mentioned when entering the closet or when looking around.
When necessary, the carpet and the trapdoor will respond to “examine floor”. The floor object will check whether carpet or trapdoor are visible and if not, it will make sure (through nomatch()) that the examine verb prints the default message.

```
$OBJECT o_floor
 DESCRIPTIONS
  d_sys "the floor"
 CONTAINED in l_closet
TRIGGERS
  "x [o_floor] " -> t_exa
  t_entrance  # don't call common t_entrance
    agree()
  t_exa
    if owns(l_closet, o_carpet) OR not(testflag(o_trapdoor.f_hidden)) then
      # do nothing, carpet and/or trapdoor will print a message
      agree()
```

```
    else
      # let verb print default message
      nomatch()
END_OBJ
```

Next are the carpet and the trapdoor.

Carpet

The carpet hides the trapdoor. The sides of the carpet are glued to the floor. To reveal the trapdoor, the player has to cut the sides of the carpet with the glass fragment. After cutting the carpet we don't want it to lay around, so we move it into the player's inventory.

```
$OBJECT o_carpet
 DESCRIPTIONS
  d_sys "the old carpet"
  d_longdescr    "The carpet doesn't seem very expensive. It just /
             about covers the floor. On a closer examination, it /
             turns out that its sides are glued to the floor."
  d_shortdescr  "On the floor is an old carpet."
  d_cut "You use the [o_fragment] to cut along the glued /
             sides of the carpet. You grab the middle part /
             of the carpet that now is no longer attached to /
            the floor and lift it. Removing the carpet /
             reveals a trapdoor in the floor!."
  d_no_move    "The carpet won't move. On closer examination /
             you find that its edges are glued to the floor."
  d_exa_moved"It's just an old carpet with the edges cut off /
             by a sharp object."
 CONTAINED on o_floor
 FLAGS
   f_takeable     = 1
   f_moveable   = 1
   f_cut = 0
 TRIGGERS
   "inventory"    -> t_i
   "examine [o_carpet]"-> t_exa
   "examine [o_floor]"   -> t_exa
   "lift [o_carpet]"         -> t_move
   "take [o_carpet]"       -> t_move
   "move [o_carpet]"     -> t_move
   "cut [o_carpet] with [o_fragment]"    -> t_cut
 t_exa
   if not(testflag(f_cut)) then
     nomatch()
   else
     printcr(d_exa_moved)
 t_move
   if testflag(f_cut) then
     printcr("You already cut the carpet loose.")
   else
     printcr(d_no_move)
 t_cut
```

```
  if not(testflag(f_cut)) then
    printcr(d_cut)
    setflag(f_cut)
    clearflag(o_trapdoor.f_hidden)
    move(o_carpet, o_player)
  else
    printcr("You already cut the carpet.")
  endif
  disagree()
END_OBJ
```

trapdoor

When the player opens the trapdoor while the flames are not extinguished, we only allow him three more turns in the closet before it gets too hot. We define a timer m_heat that counts down and fires after three moves

timer m_heat

```
m_heat
 init          3
 step          1
 direction     down
 interval      1
 state         stop
 trigger_at    0
 execute       l_closet.t_leave
```

We must define a local trigger t_leave with the closet object.

Situations when the timer is started/stopped/updated:

- when the player enters the closet with trapdoor open and flames not extinguished: timer started;
- when the player is in the closet and opens the trapdoor and flames not extinguished: timer started;
- when the player leaves the closet: timer stopped and set to 3 in trigger t_exit;
- when the player is in the closet and closes the trapdoor: timer stopped and set to 3.

object trapdoor

```
$OBJECT o_trapdoor
 DESCRIPTIONS
  d_sys "the trapdoor", “the trap door”
  d_longdescr   "The trapdoor is made of laminated wood. It seems large /
           enough for a person to fit through.."
  d_shortdescr  "In the middle of the floor is a trapdoor, "
  d_open        "The trapdoor gives access to the cellar. Through the open /
           trapdoor you see a stairway leading down."
 CONTAINED in l_closet
 FLAGS
  f_hidden      = 1
  f_openable    = 1
 TRIGGERS
```

```
  "examine [o_trapdoor]"          -> t_exa
  "open [o_trapdoor]"   -> t_open
  "close [o_trapdoor]"   -> t_close
  t_entrance
    if not(testflag(f_hidden)) then
      print(d_shortdescr)
      setflag(f_seenbefore)
      if testflag(f_open) then
        printcr("which is open.")
        # player cannot see the flames
        if not(testflag(o_flames.f_extinguished)) then
          printcr(o_flames.d_flames)
          starttimer(m_heat)  # will count down to 0
        endif
      else
        printcr("which is closed.")
  t_exit
    if testflag(o_flames.f_extinguished) then
      # stop and reset the heat timer
      stoptimer(m_heat)
      m_heat = 3
  t_open
    setflag(f_open)
    print(d_open)
    if not(testflag(o_flames.f_extinguished)) then
      starttimer(m_heat)
      printcr(o_flames.d_flames)
    else
      printcr("")
  t_close
    if not(testflag(o_flames.f_extinguished)) then
      printcr("It's less hot now. This feels much better.")
      stoptimer(m_heat)
      m_heat = 3
    else
      printcr("closed.")
    endif
    clearflag(f_open)
END_OBJ
```

And we also need a trigger t_leave that we will code in location l_closet. Why in l_closet and not in the trapdoor? Well, both are possible, we chose l_closet because leaving seems like a location thing.

new version of l_closet

```
$LOCATION l_closet
 DESCRIPTIONS
  d_sys "the closet"
  d_longdescr   "You are in a dark closet below the staircase. To the west is /
            the closet door, which is closed."
  d_shortdescr  "Closet"
  d_leave         "\nThe heat is getting too much for you. You hurry back up to the /
            stairs where it is much cooler."
 EXITS
  u -> l_halfway
 TRIGGERS
  "examine [l_closet]" -> o_player.t_look
  t_entrance
    printcr(d_shortdescr)
    printcr(d_longdescr)
    if not(testflag(o_trapdoor.f_hidden)) then
      printcr("Visible exits are up and down.")
    else
      printcr("The only visible exit is up.")
    endif
  t_leave
    # timer m_heat has fired
    stoptimer(m_heat)
    m_heat = 3
    printcr(d_leave)
    move(o_player, u)
    printcr("")
    printcrbold(l_halfway.d_shortdescr)
END_LOC
```

flames

```
$OBJECT o_flames
 DESCRIPTIONS
  d_sys "the flames", "the fire"
  d_longdescr   "Because of the heat you cannot get close enough for
          a good examination."
  d_shortdescr  ""  # flame entrance printed by the trapdoor
  d_flames      "A tremendous heat is coming through the open trapdoor. /
          You look down and see a dark red glow deep down in /
          the cellar."
  d_extinguish  "As soon as the water touches the flames, you hear a loud hissing /
          sound, followed by the appearance of lots of steam. After a /
          while, the hissing gets less until it completely stops. The fire /
           has died.\nIt seems safe to go down into the cellar now."
 CONTAINED in l_cellar
 FLAGS
  f_extinguished = 0
 TRIGGERS
  "examine [o_flames]" -> t_exa
  "extinguish [o_flames]"          -> t_extinguish
  t_entrance
   agree()
  t_extinguish
   printcr("It's up to you to find a way how to do that.")
END_OBJ
```

water tap

The water tap is in the bedroom. The tap can be opened and closed. “Turn tap” checks the current position and then does the opposite.

When the following prerequisites have been fulfilled when opening the tap:

- trapdoor is open;
- drain pipe in closet is cut with the hacksaw;
- fire is not extinguished.

The fire in the cellar will be extinguished.

If the trapdoor is closed but the drain pipe has been cut, there will be water in the north hallway, pouring from under the closet door.

```
$OBJECT o_tap
 DESCRIPTIONS
  d_sys "the tap"
  d_longdescr   "It's a tap for cold water."
  d_shortdescr  ""  # printed in t_entrance from sink.
  d_open        "As you turn the tap to open it, water starts /
          pouring into the sink."
  d_extinguish  "After a little while, you faintly here a hissing /
```

```
            sound, coming from somewhere below."
CONTAINED in l_bedroom
FLAGS
  f_openable = 1  # for open prologue
TRIGGERS
  "examine [o_tap]"      -> t_exa
  "open [o_tap]"         -> t_open
  "close [o_tap]"        -> t_close
  "turn [o_tap]"-> t_turn
  "turn on [o_tap]"      -> t_open
  "turn off [o_tap]"     -> t_close
  t_entrance
    # tap is handled by sink, because we want
    # to execute the sink t_entrance first
    agree()
  t_exa
    if testflag(f_open) then
      printcr("Water is pouring out of the tap into the sink.")
    else
      printcr("The tap is closed.")
    endif
    disagree()
  t_open
    if testflag(f_open) then
      printcr("The water is already running.")
    else
      setflag(f_open)
      clearflag(o_water_bedroom.f_hidden)
      if testflag(o_drain_pipe_closet.f_cut) then
        clearflag(o_water_closet.f_hidden)
        if not(testflag(o_trapdoor.f_open)) then
          # put water in the hallway north
          clearflag(o_water_hall_n.f_hidden)
        endif
      endif
      printcr(d_open)
      if not(testflag(o_flames.f_extinguished)) and
        testflag(o_trapdoor.f_open) and testflag(o_drain_pipe_closet.f_cut) then
        setflag(o_flames.f_extinguished)
        printcr(d_extinguish)
      endif
    endif
  t_close
    if not(testflag(f_open)) then
      printcr("It's already closed.")
    else
      clearflag(f_open)
      setflag(o_water_bedroom.f_hidden)
      setflag(o_water_closet.f_hidden)
      # water in hallway north remains
      printcr("The waterflow stops when you close the tap.")
  t_turn
```

```
    if testflag(f_open) then
      if not(trigger(t_close)) then
        disagree()
      endif
    else
      if not(trigger(t_open)) then
        disagree()
      endif
    endif
END_OBJ
```

sink object

The sink is there because we need the drain pipe. It's a scenery object.

```
$OBJECT o_sink
 DESCRIPTIONS
  d_sys "the sink"
  d_longdescr   "The sink is connected to a drain pipe, which disappears /
          into the floor."
  d_shortdescr  "There is a sink mounted to the wall. Above the sink /
          is a water tap."
 CONTAINED in l_bedroom
 TRIGGERS
  "examine [o_sink]" -> t_exa
END_OBJ
```

We're almost there. All we must do now is describe water objects to make the game more realistic. We want to allow the player to refer to the water when he opens the tap. There are three locations where the player can refer to the water: in the bedroom, in the closet and in the north hallway when the waters comes from under the closet door when the trapdoor is closed.

And of course, when we have water, we must also have a "drink" verb.

water in bedroom

When the tap is closed the water is hidden.

```
$OBJECT o_water_bedroom
 DESCRIPTIONS
  d_sys "the water"
  d_longdescr   "Just plain ordinary water."
  d_shortdescr  "Water is running from the tap into the sink."
 CONTAINED in l_bedroom
 FLAGS
  f_hidden        = 1
  f_takeable      = 1
 TRIGGERS
  "examine [o_water_bedroom]"          -> t_exa
  "get [o_water_bedroom]"      -> t_get
  "drink [o_water_bedroom]"   -> t_drink
  t_get
    printcr("You have nothing with you that can hold the water.")
  t_drink
    printcr("That's refreshing! You didn't realize you were thirsty.")
END_OBJ
```

water in closet

```
$OBJECT o_water_closet
 DESCRIPTIONS
  d_sys "the water"
  d_longdescr   "Just plain ordinary water."
  d_shortdescr  "" # printed by drain pipe
  d_no_drink    "It's better not to drink from the floor. if /
            you are thirsty, better go to the tap in /
            the bedroom for some fresh water."
 CONTAINED in l_closet
 FLAGS
  f_hidden       = 1
  f_takeable     = 1
 TRIGGERS
  "examine [o_water_closet]" -> t_exa
  "get [o_water_closet]"     -> t_get
  "drink [o_water_closet]"   -> t_drink
  t_entrance
    agree()   # handled by closet
  t_get
    printcr("You have nothing with you that can hold the water.")
  t_drink
    printcr(d_no_drink)
END_OBJ
```

water in hallway north

```
$OBJECT o_water_hall_n
 DESCRIPTIONS
  d_sys "the water"
  d_longdescr   "Just plain ordinary water."
  d_shortdescr  "From underneath the closet door, water /
            is coming into the hallway."
  d_no_drink    "It's better not to drink from the floor. if /
            you are thirsty, better go to the tap in /
            the bedroom for some fresh water."
 CONTAINED in l_hallway_north
 FLAGS
  f_hidden       = 1
  f_takeable     = 1
 TRIGGERS
  "examine [o_water_hall_n]"   -> t_exa
  "get [o_water_hall_n]"          -> t_get
  "drink [o_water_hall_n]"        -> t_drink
  t_get
    printcr("You have nothing with you that can hold the water.")
  t_drink
    printcr(d_no_drink)
END_OBJ
```

Verb drink

```
$VERB drink
 "drink"
   printcr("What do you want to drink?")
   getsubject()
 "drink [o_subject]"
   printcr("[the] [o_actor] cannot drink [the] [o_subject].")
 DEFAULT
   printcr("I only understood you as far as wanting to drink something.")
ENDVERB
```

drain pipe in bedroom

The drain pipe in the bedroom is sort of scenery. It is used to help the player make the link between the drain pipe in the closet and the bedroom and to deduct that he should cut the pipe in the closet and turn on the water to extinguish the flames.

We have a rejection message in case the player tries to saw this drain pipe.

```
$OBJECT o_drain_pipe_bedroom
 DESCRIPTIONS
  d_sys "the drain pipe"
  d_longdescr   "The drain pipe emerges from the sink and disappears in /
            the floor."
  d_shortdescr  "Attached to the wall is a drain pipe."
  d_no_cut      "It makes little sense to cut the drain pipe here."
 CONTAINED in l_bedroom
 TRIGGERS
  "examine [o_drain_pipe_bedroom]"   -> t_exa
  "cut [o_drain_pipe_bedroom] with [o_hacksaw]"       -> t_cut
  t_cut
    printcr(d_no_cut)
END_OBJ
```

drain pipe in closet

```
$OBJECT o_drain_pipe_closet
 DESCRIPTIONS
  d_sys "the drain pipe"
  d_longdescr   "The drainpipe comes down where the ceiling meets /
            the west wall, goes vertically down the west /
            wall and disappears in the floor."
  d_shortdescr  "Attached to the wall is a drain pipe."
  d_cut "About halfway up the wall, the pipe has been cut."
  d_cut_again   "You try to cut the pipe (again), but the hacksaw has /
          become blunt after you used it the first time."
  d_pour          "Water pours out of the upper half of the broken pipe /
            on the floor, "
  d_pour_to_hallway    "where it disappears under the closet door into the hallway."
  d_pour_in_cellar       "through the open trapdoor straight into the cellar."
 CONTAINED in l_closet
 FLAGS
  f_cut = 0  # not yet cut.
 TRIGGERS
  "examine [o_drain_pipe_closet]"       -> t_exa
  "cut [o_drain_pipe_closet] with [o_hacksaw]"-> t_cut
  t_entrance
    print(d_shortdescr)
    if testflag(f_cut) then
      print(d_cut))
      if testflag(o_tap.f_open) then
        print(d_pour)
        if testflag(o_trapdoor.f_open) then
          printcr(d_pour_in_cellar)
        else
          printcr(d_pour_to_hallway)
        endif
      endif
```

```
    else
      if testflag(o_tap.f_open) then
        printcr("You hear water running through the pipe.")
  t_exa
    if not(testflag(f_cut)) then
      printcr(d_longdescr)
    else
      printcr(d_cut)
        if testflag(o_tap.f_open) then
          print(d_pour)
          if testflag(o_trapdoor.f_open) then
            printcr(d_pour_in_cellar)
          else
            printcr(d_pour_to_hallway)
          endif
        endif
    endif
  t_cut
    if not(testflag(f_cut)) then
      setflag(f_cut)
      setflag(o_hacksaw.f_worn)
      printcr("You cut the pipe about halfway above the floor.")
      if testflag(o_tap.f_open) then
        print(d_pour)
        clearflag(o_water_closet.f_hidden)
        if not(testflag(o_trapdoor.f_open)) then
          printcr(d_pour_to_hallway)
        else
          printcr(d_pour_in_cellar)
          printcr(o_flames.d_extinguish)
          move(o_flames, l_storage)
        endif
      endif
    else
      printcr(d_cut_again)
    endif
END_OBJ
```

End of part 3

This ends part 3 of the tutorial. We now have a complete playable story. It's not the most exiting story, but the purpose of this tutorial is to show how to make an XVAN story, it's not a writing course.

Everything we've done until now is in files part3-end.lib and part3-end.xvn. To make a playable game file, run the compiler and enter part3-end.xvn as the story file name. Name the output file 'out.dat'. The output file may have any name, but if you want to use the Glk Interpreter, it must be called out.dat. The compiler will generate the output file that can be played using the interpreter. How to start the compiler and interpreter for different operating systems can be found in the XVAN installation and user guide.

In the remainder of this tutorial are two optional parts. Optional meaning that they are not necessary because we have a working story after part 3.

Part 4 goes into the look and feel. It changes background and text colors to white on blue and it makes use of the status window for the Glk version of the interpreter.

Part 5 demonstrates how to build some intelligence into verbs to parse ambiguous user input without asking the user for further clarification.