The Alan Cookbook

by

Anssi Räisänen

Version 2
November 24, 2012

Thanks to Robert DeFord for the PDF version as well as various ideas and suggestions.

This document provides coding examples that show how to implement many of the things commonly required to create interactive fiction using the ALAN language. Since the examples have minimal explanations, you should probably read through the first three chapters in the Alan Adventure Language Reference Manual prior to adapting them for use in your own source code.

An (L) after the line indicates that the standard library must be imported by your source code in order for the example to work.

LOCATIONS

Implement a basic location

```
THE kitchen ISA LOCATION
  DESCRIPTION "This is the kitchen of your house."
  EXIT west TO livingroom.
END THE kitchen.
```

(Also just "`END THE.`" is possible for brevity's sake. This applies to all instances - locations, actors and objects.)

Implement an indoor location (L)

```
THE kitchen ISA ROOM
  DESCRIPTION "This is the kitchen of your house."
  EXIT west TO livingroom.
END THE.
```

- when you declare a location a ROOM, it will automatically have a floor, a ceiling and walls.

Implement an outdoor location (L)

```
THE garden ISA SITE
  DESCRIPTION "This is your garden."
  EXIT 'in' TO house.
END THE garden.
```

- when you declare a location a SITE, the ground and the sky will automatically be present

Implement a dark location (L)

All locations are lit by default. All dark locations are NOT lit by default (this is declared in the standard library.)

```
THE basement ISA DARK_LOCATION
  EXIT up TO kitchen.
END THE basement.
```

All actions requiring vision are disabled in a dark location.

If you intend to make a dark_location lighted at some point in the game, add a description:

```
THE basement ISA DARK_LOCATION
  DESCRIPTION "Cobwebs and useless junk are the only things you see here."
  EXIT up TO kitchen.
END THE.
```

This description will be shown only when the dark_location is lit.

Make a dark location lit (L)

```
THE basement ISA DARK_LOCATION
  DESCRIPTION "Cobwebs and useless junk are the only things you see here."
  EXIT up TO kitchen.

  VERB switch_on
    CHECK basement IS NOT lit
      ELSE "The lights are already on."
    DOES ONLY
      IF obj = lightswitch
        THEN "You turn on the lights."
          MAKE basement lit.
        ELSE "That's not possible."
      END IF.
  END VERB.
END THE basement.
```

Make a location that starts up as lit but gets dark later (L)

```
THE basement ISA DARK_LOCATION
  IS lit.
  ENTERED
    SCHEDULE light_off AT basement AFTER 3.
    DESCRIPTION "The flickering light here might go out at any moment."
END THE basement.


EVENT light_goes_off
  "The light goes out!"
  MAKE basement NOT lit.
END EVENT.
```

Make all dark locations in a game lighted at the same time

```
THE main_power_switch ISA DEVICE AT lab
   ...
  VERB switch_on
    DOES ONLY
      FOR EACH dl ISA DARK_LOCATION
        DO
        MAKE d1 lit.
      END FOR.
  END VERB.
...
END THE.
```

Implement a location with no floor/ground, a treetop in this example.

- just declare the location a normal LOCATION (and not e.g. a SITE).
- mainly it is important to ensure that every dropped item ends up on the ground below instead of in the treetop:

```
THE treetop ISA LOCATION
  EXIT down TO under_the_tree.
  ...
  VERB drop
    DOES ONLY LOCATE obj AT under_the_tree.
      "You drop" SAY THE obj. "to the ground."
  END VERB.
...
END THE treetop.

THE under_the_tree ISA LOCATION
  NAME Under 'the' tree
  EXIT up TO treetop.
END THE under_the_tree.
```

Implement only a part of a location (e.g. The West End Of The Great Hall)

Say you have a big location split in two; e.g. 'The West End Of The Great Hall' and 'The East End Of The Great Hall'. There are two ways this can be implemented:

1) Just implement them as independent locations that stand next to each other:

```
THE west_hall ISA LOCATION
  NAME 'West' 'End' 'Of' 'The' Great Hall
  DESCRIPTION "This is a vast hall continuing to the east."
  EXIT east TO east_hall.
END THE west_hall.

THE east_hall ISA LOCATION
  NAME 'East' 'End' 'Of' 'The' Great Hall
  DESCRIPTION "This is a vast hall continuing to the west."
  EXIT west TO west_hall.
END THE east_hall.
```

- With this method, the objects or actors in the other end of the hall are not in scope.

2) Make one metalocation, great_hall, and nest the eastern and western parts into it:

```
THE great_hall ISA LOCATION
END THE.

 THE west_hall ISA LOCATION AT great_hall
  NAME 'West' 'End' 'Of' 'The' Great Hall
  DESCRIPTION "This is a vast hall continuing to the east."
  EXIT east TO east_hall.
END THE west_hall.

THE east_hall ISA LOCATION AT great_hall
  NAME 'East' 'End' 'Of' 'The' Great Hall
  DESCRIPTION "This is a vast hall continuing to the west."
  EXIT west TO west_hall.
END THE east_hall.
```

- With this method, all objects or actors in the metalocation (great_hall) will be available alsoin the eastern and western halls (useful for example if you have objects that are present in both ends of the hall, e.g. a red carpet).
(An object implemented into west_hall won't, however, be present in the east_hall, and vice versa.)

Implement nested locations

```
THE garden ISA LOCATION
END THE garden.

THE fruit_trees ISA SCENERY AT garden
END THE garden.

THE by_the_pond ISA LOCATION AT garden
END THE by_the_pond.
```

Here, the fruit trees will be in scope in the by_the_pond area as well.

Implement groups of locations ("areas" or "regions")

Use the nested locations feature. Here, the locations First Street, Second Street and Third Street belong to the town area and the Highway is outside the town.

```
THE town ISA LOCATION -- the area, or the region
END THE town.

THE first_street ISA LOCATION AT town
  NAME first street
  EXIT north TO second_street.
END THE first_street.

THE second_street ISA LOCATION AT town
  NAME second street
  EXIT south TO first_street.
  EXIT west TO third_street.
END THE second_street.

THE third_street ISA LOCATION AT town
  NAME third street
  EXIT east TO second_street.
  EXIT north TO highway.
END THE third_street.

THE highway ISA LOCATION -- not in the town!
  EXIT south TO third_street.
END THE highway.
```

Make certain things happen only when the hero is in a certain area/region

Change an attribute of the hero when he is in a certain area or region, and check the value of this attribute:

1)

```
THE highway ISA LOCATION
  EXIT south TO third_street
  DOES MAKE hero in_town.
END THE highway.
```

2)

```
IF hero IS in_town
  THEN "..."
END IF.
```

3)

```
VERB windowshop
    CHECK hero IS in_town
     ELSE "There are no shops here!"
     DOES "You admire the items for sale in the shop windows. If only
you had the money."
END VERB.
```

Connect locations to each other using exits

```
THE kitchen ISA LOCATION
  DESCRIPTION "This is your kitchen."
  EXIT west TO livingroom.
END THE kitchen.

THE livingroom ISA LOCATION
  DESCRIPTION "This is your livingroom."
  EXIT east TO kitchen.
END THE livingroom.
```

- exits must be manually programmed both ways. When you declare an exit from one location to another, there will be no exit automatically back to the first location.

Make exits do other things besides relocating the hero (change an attribute, display a message etc.)

```
THE bedroom ISA LOCATION
  DESCRIPTION "This is your bedroom."
  EXIT north TO corridor
    DOES "You decide to dress up quickly and make your bed before leaving
your bedroom."
      LOCATE pyjamas IN bed.
      LOCATE suit IN hero.
      LOCATE shoes IN hero.
      MAKE bed made.
  END EXIT.
END THE bedroom.
```

Implement restricted exits

```
THE bedroom ISA LOCATION
  DESCRIPTION "This is your bedroom. A closet lies to the east."
  EXIT east TO closet
    CHECK closet_door IS NOT closed
       ELSE "The door to the closet is presently closed."
  END EXIT.
END THE bedroom.
```

Here, `EXIT` statement will be carried out only if the closet door is not closed.
No `DOES` statement is needed; if the check is passed, the `EXIT` will automatically relocate the hero.

Implement fake exits

```
THE bedroom ISA LOCATION
  DESCRIPTION "This is your bedroom. A closet lies to the east."
  EXIT east TO nowhere
    CHECK "The closet is closed and locked. There's no way in this game for
you to get in there."
  END EXIT.
END THE bedroom.

THE nowhere ISA LOCATION
END THE nowhere.
```

An unconditional `CHECK` (without any `ELSE` or `DOES` following) will block the `EXIT` from executing.

Implement random exits

In this example, typing 'east' in the main cave will place the hero in one of three alternative locations at random:

```
THE main_cave ISA LOCATION
  DESCRIPTION "There seem to be some small passages to the east."
  EXIT east TO nowhere
    CHECK
      DEPENDING ON RANDOM 1 TO 3
        = 1 THEN LOCATE hero AT cave1.
        = 2 THEN LOCATE hero AT cave2.
        = 3 THEN LOCATE hero AT cave3.
      END DEPEND.
END EXIT.

END THE main_cave.

THE cave1 ISA LOCATION
END THE cave1.

THE cave2 ISA LOCATION
END THE cave2.

THE cave3 ISA LOCATION
END THE cave3.
```

Implement new directions (e.g. shipboard directions)

Just put any arbitrary direction name after EXIT:

```
THE cabin ISA ROOM
  EXIT aft TO storage.
  EXIT starboard TO corridor.
END THE cabin.
```

Or

```
THE whoowhoo ISA LOCATION
  EXIT anywhichway TO haahaa
```

```
END THE whoowhoo.
```

Implement doors between locations (L)

All DOORS are by default closeable, closed, lockable and NOT locked.

A door object will be automatically described as being opened or closed when examined.

```
THE kitchen ISA LOCATION
   ...
  EXIT west TO livingroom
    CHECK kitchen_door IS NOT closed
      ELSE "The door to the living-room is closed."
  END EXIT.
END THE kitchen.

THE kitchen_door ISA DOOR AT kitchen
  VERB open
    DOES MAKE livingroom_door NOT closed. -- the same door but from the
other side
  END VERB.

  VERB close
    DOES MAKE livingroom_door closed. -- the same door but from the other
side
  END VERB.
END THE kitchen_door.

THE livingroom ISA LOCATION
   ...
  EXIT east TO kitchen
    CHECK livingroom_door IS NOT closed
      ELSE "The door to the living-room is closed."
  END EXIT.
END THE livingroom.

THE livingroom_door ISA DOOR AT livingroom
  VERB open
    DOES MAKE kitchen_door NOT closed. -- the same door but from the other
side
  END VERB.

  VERB close
    DOES MAKE kitchen_door closed. -- the same door but from the other side
  END VERB.
END THE livingroom_door.
```

Implement exits that weren't there before (e.g. when a secret door is revealed)

```
THE cave ISA LOCATION
  ...
  EXIT out TO outside_cave.
    EXIT west TO treasure_chamber -- through a secret door
      CHECK secret_door AT cave
        ELSE "You can't go that way."
  END EXIT.
END THE cave.
```

```
THE secret_door ISA DOOR
END THE.
```

Make something happen when a location is entered

```
THE room1 ISA LOCATION
  ENTERED
    SCHEDULE phone_ringing AT room1 AFTER 0.
END THE room1.
```

Use `ENTERED`. Note that if a non-player character is moving in the game map, `ENTERED` will be executed also when this character enters the location.
That's why it is sometimes handy to check if the executing actor is the hero or somebody else.
This is achieved through using `CURRENT ACTOR`:

```
THE room1 ISA LOCATION
  ENTERED
    IF CURRENT ACTOR <> hero
      THEN SCHEDULE notifying_message AT room1 AFTER 0.
    END IF.
END THE room1.

EVENT notifying_message
  IF CURRENT ACTOR = lisa
    THEN "Lisa enters the room!"
  ELSIF CURRENT ACTOR = tom
    THEN "Tom walks in!"
  ELSIF...
  END IF.
END EVENT.
```

(or just:

```
EVENT notifying_message
  SAY CURRENT ACTOR. "enters the room!"
END EVENT.)
```

Not display a location name when entering or LOOKing

```
THE kitchen ISA LOCATION
  NAME ''
  DESCRIPTION “Blah blah…”
END THE kitchen.
```

Not display a location description when entering or LOOKing

Just leave the description out altogether:

```
THE kitchen ISA LOCATION
END THE kitchen.
```

Not display either a location name or description when entering or LOOKing

```
THE kitchen ISA LOCATION
  NAME ''
END THE kitchen.
```

Display a message before the location name and description upon entering a location

Put the message into the DOES part of the EXIT of the previous location:

```
THE loc1 ISA LOCATION
  EXIT east TO loc2
    DOES "When you enter, a strange odor catches your attention."
  END EXIT.
END THE loc1.
```

Display a message after the location description upon entering a location

Have an event trigger when the location is entered:

```
THE livingroom ISA LOCATION
  DESCRIPTION "Blah blah..."
  ENTERED
    SCHEDULE livingroom_afterdesc AFTER 0.
END THE livingroom.

EVENT livingroom_afterdesc
  "$pThis message will be shown after the description of the living-room."
END EVENT.
```

Display a message after the location name but before the location description upon entering a location

Just put the message in the location description itself:

```
THE kitchen ISA LOCATION
  DESCRIPTION
  "(Wow, what an ingenious name for this location. Before going to the
description part,
   you taste the word in your mouth for a while.)$p"
  "You are in the kitchen."
END THE kitchen.
```

If you wish to have the special message displayed e.g. only the first time the kitchen is visited, you can add the following code if you are using the standard library:

```
DESCRIPTION
  IF visited OF kitchen = 1
    THEN "(Wow, what..."
  END IF.
  "You are in the kitchen."
```

Display a message upon exiting a location

Put the message into the DOES part of an EXIT:

```
      THE loc1 ISA LOCATION
        EXIT east TO loc2
          DOES "When you leave, you have the feeling that you forgot to pick up
something crucial."
        END EXIT.
      END THE loc1.
```

Vary the location description e.g. so that the first-time description is different from the subsequent times (L)

Use the 'visited' or 'described' attributes.

1)

```
      THE kitchen ISA LOCATION
         DESCRIPTION
          "You are in the kitchen."
          IF visited OF kitchen = 1 -- (= your first time here)
            THEN "This is your first time here."
            ELSE "You remember you've been here before."
          END IF.
      END THE kitchen.
```

2)

```
      THE library ISA ROOM
        DESCRIPTION
          IF described OF library = 1 -- (= the first-time description)
            THEN "There is an old man reading at a table in one of the
corners."
            ELSE "The old man keeps on reading at his table."
          END IF.
      END THE library.
```

Change the attribute of a location

```
      THE bedroom ISA LOCATION
        IS NOT cleaned.

        VERB clean
          CHECK bedroom IS NOT cleaned
            ELSE "The bedroom is already cleaned."
          DOES "You clean the bedroom."
            MAKE bedroom cleaned.
        END VERB.
      END THE bedroom.
```

Check an attribute of the current location

```
      VERB read
        CHECK CURRENT LOCATION IS lit
      ELSE "It's too dark to see!"
    DOES "You read."
  END VERB.
```

Keep track of whether or how many times a location has been visited (L)

```
  THE king ISA ACTOR
    ...
    VERB ask
        WHEN act
        IF topic = treasure_chamber
          THEN
            IF visited OF treasure_chamber = 0
              THEN "You are not supposed to know anything about the treasure
                chamber - you haven't found it yet."
              ELSE """Just take what you want from the chamber"", the king
smiles."
            END IF.
        ...
        END IF.
     END VERB.
  END THE king.
```

Keep track of how many times a location has been described (L)

```
  THE library ISA ROOM
    DESCRIPTION
      IF described OF THIS = 1
        THEN "There is an old man reading at a table in one of the corners."
      ELSIF described OF THIS < 5
        THEN "The old man keeps on reading at his table."
      ELSE "The old man seems to never get tired of reading."
      END IF.
  END THE.
```

Make a verb behave differently in a given location (e.g. >jump The ceiling is too low here.)

```
  THE basement ISA LOCATION
    DESCRIPTION ...
    VERB jump
      DOES ONLY "The ceiling is too low here."
    END VERB.

    VERB take
      DOES ONLY "There is nothing worth taking among the old junk here."
    END VERB.
  END THE basement.
```

Implement a location or situation where all or some verbs and commands are restricted so that they display the same message

(e.g. >jump But you're tied up! >take keys But you're tied up! >attack guard But you're tied up!)
Use the extension "Restricted verbs" available on the ALAN website.

Allow the player to refer to the location (e.g. 'enter kitchen', 'search basement')

You can't refer to the location instance directly in a player command. Make a reference object instead:

```
 THE kitchen_object ISA OBJECT AT kitchen
  NAME kitchen

  VERB examine
    DOES ONLY "A modern kitchen with many appliances."
  END VERB.

  VERB 'exit'
    DOES ONLY LOCATE hero AT living_room.
  END VERB.
  ...
END THE kitchen_object.
```

Check the location of an object or actor

1)

```
IF hero AT treasure_chamber
  THEN ...
END IF.
```

2)

```
IF ball AT tom -- at the same location as Tom
  THEN ...
END IF.
```

3) In verb checks:

```
CHECK hero AT treasure_chamber
  ELSE ...
DOES...
```

4)

```
IF LOCATION OF ball IS NOT lit
  THEN ...
END IF.
```

OBJECTS

Implement a basic object

```
THE ball ISA OBJECT AT room1
  NAME red ball
  DESCRIPTION "A red ball lies to one side."

  VERB examine
    DOES ONLY "It is an ordinary red ball."
  END VERB.
```

```
 END THE ball.
```

Have the default object description show up after the location description when looking or entering

(e.g. "There is a ball here.")

```
THE ball ISA OBJECT AT garden
END THE ball.
```

- leaving any description of your own out altogether will result in the object being described after the location
- description with the default sentence "There is a [object] here."

Have your own object description, instead of the default, show up after the location description when looking or entering

Usually, an object is described in the location with the sentence "There is a (x) here." after the location description.

You have three other alternatives:

1) edit this message
2) remove it altogether (and include the object in the location description in your own words)
3) edit the default runtime message that mentions the object in the location. (However, this will affect all objects in the game.)

As Follows:

1) To just edit "There is a (x) here.", you can do

```
THE ball ISA OBJECT
  DESCRIPTION "A red ball lies on the floor."
END THE ball.
```

This sentence will then display after the location description, instead of "There is a ball here."
(This sentence will also display if the ball is dropped in any other location, including outdoors; that's why it is a good idea to take into account all possible alternatives and provide multiple descriptions as needed; or then make the description generic enough to suit all possible locations.)

2) To remove this description, do like this:

```
THE ball ISA OBJECT
  DESCRIPTION ""
END THE ball.
```

Leave the description empty. You can then include the object in the location description itself, using your own words. (Remember to remove the object from the location description if it is picked up.)

3) To edit the default runtime message:

```
MESSAGE SEE_START: "You see $01"
```

Yields
```
"You see a ball here."
(instead of the default "There is a ball here.")
```

This will, however, affect all objects in the game the same way.

Implement an opaque container object

```
THE closed_box ISA OBJECT AT storage
  NAME box
  OPAQUE CONTAINER
END THE closed_box.

THE ball ISA OBJECT IN closed_box
END THE ball.
```

The ball won't be visible to the player, and it can't be interacted with.

Implement a transparent container object

```
THE glass_bowl ISA OBJECT AT kitchen
  NAME glass bowl
  CONTAINER
END THE glass_bowl.

THE apple ISA OBJECT IN glass_bowl
END THE apple.
```

Something in a normal (transparent) container object won't be listed automatically when the container is described (after `LOOK` or `EXAMINE`) - only the verb `LOOK IN` will work. If you wish that the contents of a container object are automatically listed after `LOOK` and `EXAMINE`, use the standard library and declare the container a `LISTABLE_CONTAINER`:

```
THE glass_bowl ISA LISTABLE_CONTAINER AT kitchen
END THE.
```

Implement open and closed containers (L)

1)
```
THE box ISA OBJECT AT basement
  CONTAINER
  IS closable. IS closed.
 ...
END THE box.
```
2)
```
THE jar ISA LISTABLE_CONTAINER IN kitchen
  IS closable. IS NOT closed.
...
END THE jar.
```

Make opaque containers show their contents when opened

```
THE box ISA LISTABLE_CONTAINER AT basement
  OPAQUE
  IS closable. IS closed.
 ...

  VERB open
    DOES ONLY MAKE box NOT closed. MAKE box NOT OPAQUE.
  END VERB.
END THE box.
```

(This is handled automatically by the standard library.)

Edit the description of what a container is holding

```
THE sack ISA OBJECT
  CONTAINER
    HEADER "Your sack currently contains"
    ELSE "Your sack is currently empty."
    ...
END THE sack.
```

The `HEADER` part of an instance's `CONTAINER` section replaces the default runtime message `CONTAINS_START` for that instance.
The `EMPTY` part of an instance's `CONTAINER` section replaces the default runtime message `EMPTY_HANDED` for that instance.

Limit how much a container can hold

1)

```
THE sack ISA OBJECT
  CONTAINER
    LIMITS
      COUNT 10
        ELSE "The sack is already full."
END THE sack.
```

2)

```
THE sack ISA OBJECT
  CONTAINER
    LIMITS
      weight 30
        ELSE "You can't carry anything more."
END THE sack.
```

The standard library defines that every object has the numerical weight attribute 5 and every actor has the weight attribute 50.

Containers are not limited in any way in the standard library. The game author must set the limits.

The library only limits how much the hero can carry at one time (= weight 50).

Check how much a container is currently holding

```
      IF COUNT ISA OBJECT, IN sack = 5
        THEN "Bingo! The sack now contains 5 items. You now have enough equipment
to embark on your adventure."
      END IF.
```

Report how much a container is currently holding

```
      THE sack ISA OBJECT
        CONTAINER
        ...

        VERB examine
          DOES ONLY "The sack currently contains" SAY COUNT ISA OBJECT, IN sack.
"items."
        END VERB.
      END THE sack.
```

List the contents of container objects in the hero's inventory

Edit the command 'inventory' in the file 'verbs.i':

```
      VERB inventory
        DOES LIST hero.
        IF box IN hero
          THEN LIST box.
        END IF.
        IF bag IN hero
          THEN LIST bag.
        END IF.
      END VERB.
```

Control the order in which objects are listed when taking inventory or describing the contents of a container

1) If you wish to have the initial contents of a container listed in a certain order, first locate the contents in the container by using `INITIALIZE` and `LOCATE`:

```
      THE folder ISA OBJECT
         CONTAINER
          INITIALIZE
            LOCATE material1 IN folder.
            LOCATE material2 IN folder.
            LOCATE material3 IN folder.
      END THE folder.

      THE material1 ISA OBJECT
      END THE.

      THE material2 ISA OBJECT
      END THE.

      THE material3 ISA OBJECT
      END THE.
```

The three materials will be listed in this order when the contents of the folder are described.

2) Here is an example where the tools the hero is carrying are listed first, and all other items are listed thereafter. The example objects are a key, a book, a hat, a hammer, a screwdriver and a saw (the three first ones being generic objects, the three latter ones being tools). Regardless of in which order the hero picks these objects up, the tools are always listed first when taking inventory:

```
EVERY tool ISA OBJECT
END EVERY.

THE key ISA OBJECT AT room1
END THE.

THE book ISA OBJECT AT room1
END THE.

THE hat ISA OBJECT AT room1
END THE.

THE hammer ISA TOOL AT room1
END THE.

THE screwdriver ISA TOOL AT room1
END THE.

THE saw ISA TOOL AT room1
END THE.

VERB inventory
  DOES
    EMPTY hero IN rearranging_inventory.
    FOR EACH o ISA TOOL, IN rearranging_inventory
      DO
      LOCATE o IN hero.
    END FOR.
    EMPTY rearranging_inventory IN hero.
    LIST hero.
END VERB.

THE rearranging_inventory ISA OBJECT
  CONTAINER
END THE.
```

Here, the possessions of the hero are all located in another container first (= the hero's inventory is emptied to be arranged anew). After that, the tool objects are first located back in the hero's inventory, using the `FOR EACH` loop. Finally, all other objects are located back, as well.All of this is invisible to the player, and the tools will always be listed first when taking inventory.

Vary the way the hero's inventory is listed

Edit the runtime messages that control the listing of containers. Here is an example:

```
MESSAGES
  CONTAINS_COMMA: "$i$01"
```

```
   CONTAINS_AND: "$i$01"
   CONTAINS_END: "$i$01."


yields e.g.


> inventory
You are carrying
   a set of keys
   a remote control
   an umbrella
   a newspaper
   a note
   some coins.
```

Control which objects can be put into a container

Sometimes we wish to limit what can be put into a certain container. Commands like 'put book in coffee cup' should not have successful outcomes. Similarly, in a game we might have e.g. a folder object in which we should only be able to put papers or documents, but not e.g. keys or fruit. Here are a couple of examples to show how to accomplish this:

1)

```
EVERY liquid ISA OBJECT
END THE.

THE coffee ISA LIQUID AT room1
END THE.

THE book ISA OBJECT AT room1
END THE.

THE coffee_cup ISA OBJECT AT room1
   CONTAINER TAKING LIQUID.
   NAME coffee cup
END THE.
```

2)

```
EVERY document ISA OBJECT
   IS readable.
END THE.

THE newspaper_article ISA DOCUMENT AT office
END THE.

THE letter ISA DOCUMENT AT office
END THE.

THE job_application ISA DOCUMENT AT office
END THE.

THE apple ISA OBJECT AT office
END THE.
```

```
THE folder ISA OBJECT
  CONTAINER TAKING DOCUMENT.
END THE folder.
```

(`TAKING`, after `CONTAINER`, specifies which kind of objects the container can hold. After `TAKING`, any instance class or subclass may follow.)

3) Sometimes the methods described above are not sufficient. What if we need to restrict a desk drawer not accepting a chair or a suitcase, and similar cases?

a) Every object has a default weight attribute 5, defined in the standard library. We can make use of this feature by giving a high weight attribute to all objects that shouldn't go into a container:

```
THE drawer ISA OBJECT AT bedroom
  CONTAINER

  VERB put_in
      WHEN cont
      CHECK weight of obj <= 5
        ELSE "That doesn't fit into the drawer."
  END VERB.
END THE.

THE chair ISA OBJECT AT bedroom
   HAS weight 10.
END THE.

THE suitcase ISA OBJECT AT lobby
   HAS weight 10.
END THE.
```

You can also use an attribute of your own, e.g. 'size', in a similar way.

b)

```
THE drawer ISA OBJECT
    CONTAINER
  HAS allowed {diary, watch, keys}.

  VERB put_in
    CHECK obj IN allowed OF drawer
      ELSE "That doesn't belong in the drawer."
  END VERB.
END THE.
```

Allow actors inside a container

By default, actors cannot be located inside a container object. To change this, use TAKING:

```
THE bed ISA OBJECT AT bedroom
  CONTAINER TAKING ACTOR.
  ...
END THE bed.
```

Make something happen when an object is taken out of a container

EXTRACT defines what happens when something is taken out of a container, e.g. when the player types "take apple from bowl" or "take book from man". EXTRACT needs to be placed in the CONTAINER section of an instance:

```
THE man ISA ACTOR AT street
 ...
  CONTAINER
    HEADER "The man is carrying"
    ELSE "The man is empty-handed."
    EXTRACT "$p""Hey!"", the man snaps at you, ""give it back to me!"""
      MAKE man angry.
END THE man.
```

What is defined in EXTRACT will be carried out after the default outcome for the verb used.

E.g.

```
>take book from man
You take the book from the man.
"Hey!", the man snaps at you, "give it back to me!"
```

An optional CHECK in the EXTRACT section prohibits the extraction from taking place under a certain condition:

```
THE man ISA ACTOR AT street
 ...
  CONTAINER
    HEADER "The man is carrying"
    ELSE "The man is empty-handed."
    EXTRACT
      CHECK book IN man
        ELSE "It's not worth trying to snatch anything from the man at
present."
      DOES """Hey!"", the man snaps at you, ""give it back to me!"
        MAKE man angry.
END THE man.
```

Here, trying to take something from the man (when he's not carrying the book) would stop the 'take_from' verb from executing and print out the check message:

```
>take keys from man
It's not worth trying to snatch anything from the man at present.
```

An EXTRACT with a mere CHECK, with no ELSE or DOES, stops the action in all cases:

```
EXTRACT
     CHECK "It's not worth trying to snatch anything from this guy."
```

This is equivalent to:

```
THE man ISA ACTOR AT street
      ...
      VERB take_from
        WHEN holder
       DOES ONLY "It's not worth trying to snatch anything from this guy."
```

```
           END VERB.
        END THE man.
```

But EXTRACT is more flexible in that it will stop the action with all imaginable verbs (and not just 'take_from') that the player might use to get an item from the man.

Check the location of an object which might be in a container

There are two ways to check if a thing is at a certain location:

1)

```
IF ball AT room1
  THEN...
END IF.
```

If the ball is in room1, this check will find the ball whether it is e.g. on the floor, in a box or in a drawer in other words, regardless whether the ball is in a container or not, as long as it is in room1.

2)

```
IF ball DIRECTLY AT room1
  THEN ...
END IF.
```

DIRECTLY checks if the location of the object is *directly* at the checked location. That means to say that if the ball is e.g. in a box, this check will fail. Only if the ball is to be found directly in the room (e.g. "You see a red ball on the floor."), this check will be successful and the THEN part of the conditional statement will be carried out.

Take a random object out of a container

Here, the hero encounters a thief who steals one random object from the inventory:

```
        EVENT thief_appears.
          "A thief suddenly appears and takes one of your possessions! He disappears
as quickly as he emerged."
          LOCATE RANDOM IN hero In thief.
        END EVENT.
```

Empty a container object or the hero's inventory

1)

```
THE bowl ISA LISTABLE_CONTAINER AT kitchen
     ...
     VERB pour
       DOES ONLY EMPTY bowl HERE.
       "You pour the contents of the bowl onto the floor."
     END VERB.
END THE bowl.
```

(This is handled automatically by the standard library; see verbs 'pour' and 'empty' in 'verbs.i'.)

2)

```
      EVENT earthquake
        "The ground shakes violently. Everything you were carrying drops to
the ground!"
        EMPTY hero HERE.
      END EVENT.
```

Take one object out of a group of many identical ones

(E.g., take a coin out of a purse so that there are still a number of coins left in the purse, or take a spoon out of a kitchen drawer so that there are still spoons left in the drawer)

```
THE spoons ISA OBJECT IN drawer
       IS plural.

       VERB take
        DOES ONLY
          IF spoon NOT IN hero
            THEN "You take one spoon."
              LOCATE spoon IN hero.
            ELSE "You already have a spoon."
          END IF.
      END VERB.
END THE spoons.

THE spoon ISA OBJECT -- declare no location for this instance to begin with!
 ...
END THE spoon.
```

Drop a certain item when you are carrying many similar ones

(e.g. you are delivering newspapers and only deliver one newspaper at a time)

In this example, the hero is carrying ten newspapers to begin with. This example shows the crudest way to accomplish the trick; no distinction is made between the singular and the plural, and once a newspaper has been delivered, it is not possible to refer to it.

```
THE newspapers ISA OBJECT IN hero
       NAME newspaper NAME newspapers
       MENTIONED
        IF amount OF newspapers > 1
          THEN SAY amount OF newspapers. "newspapers"
          ELSE SAY "one newspaper"
        END IF.
      HAS amount 10.

      VERB examine
        DOES ONLY "You are carrying" SAY amount OF newspapers.
          IF amount OF newspapers = 1
            THEN "newspaper."
            ELSE "newspapers."
          END IF.
      END VERB.

      VERB deliver
        DOES ONLY
          IF amount OF newspapers > 1
```

```
                THEN "You deliver a newspaper."
                  DECREASE amount OF newspapers.
              ELSIF amount OF newspapers = 1
                THEN "You deliver the last newspaper. You don't have any more
left."
                  LOCATE newspapers AT nowhere.
               END IF.
          END VERB.
END THE newspapers.
```

Implement a scenery object (L)

```
THE flowerpot ISA SCENERY AT livingroom
END THE flowerpot.
```

Scenery objects can be manipulated like other objects `but` the default `EXAMINE` and `TAKE` actions report that the object is not interesting. A scenery object doesn't appear in location descriptions by default, and it can't be taken.

Implement a background object (L)

```
THE mountain ISA BACKGROUND AT valley
END THE mountain.
```

Background objects are out of reach of the player character. They don't appear in location descriptions by default. A background object differs from a scenery object in that a scenery object is within reach.

Implement a background object that is present in multiple locations simultaneously

It is possible to create an impression of an object being found in multiple locations. However, in reality this will be one and the same object all through, and not several different objects. This must be born in mind in case the object is made takeable or it is meant to be manipulated in any manner.

In this example a ceiling lamp is found in the lobby, the bedroom and the living-room of a house, but not in other locations:

First, define the area where the object(s) should be found:

```
THE lamp_rooms ISA LOCATION -- i.e. the area in which we'll nest
END THE. -- the three rooms mentioned above
```

Then define which locations belong to that area:

```
THE lobby ISA LOCATION IN lamp_rooms
 END THE.

THE bedroom ISA LOCATION IN lamp_rooms
END THE.

THE livingroom ISA LOCATION IN lamp_rooms
END THE.
```

```
Then, place the background object in the area:

THE ceiling_lamp ISA BACKGROUND IN lamp_rooms
       NAME ceiling lamp
END THE.
```

Now, a lamp is present in all of the above locations (even though in reality it is one and the same lamp). Note above that a description of the lamp will be only visible by default at the location 'lamp_rooms' (which the player doesn't visit in-game). That's why the description of the lamp must be manually included in the room descriptions of the lobby, the bedroom and the living-room. An object's being in scope means that the object can be referred to and manipulated, but its description won't appear in the location description by default.

Naturally, you could also define a scenery object or a normal object to be in several locations at once, in the same way. Note, however, that if you define a takeable object in this manner, it will disappear from the other locations once you take it in one location. Also, when manipulated (e.g. broken), the object will be affected in all of the locations it is found.

Implement a device (L)

```
THE radio ISA DEVICE AT livingroom
END THE radio.
```

A device object can be turned (=switched) on and off, if it is not broken.

Implement a piece of clothing (L)

```
THE hat ISA CLOTHING IN worn -- (= worn by the player character)
  IS headcover 2.
END THE hat.

THE coat ISA CLOTHING IN wardrobe
  IS topcover 64. botcover 64.
END THE jacket.
```

A clothing object will need an attribute from the following table:

Clothing	**Headcover**	**Topcover**	**Botcover**	**Footcover**	**Handcover**
Hat	2	0	0	0	0
Vest/bra	0	2	0	0	0
Undies/panties	0	0	2	0	0
Teddy	0	4	4	0	0
Blouse/shirt/T-shirt	0	8	0	0	0
Dress/coveralls	0	8	32	0	0
Skirt	0	0	32	0	0
Trousers/shorts	0	0	16	0	0
Sweater/pullover	0	16	0	0	0
Jacket	0	32	0	0	0
Coat	0	64	64	0	0
Socks/stockings	0	0	0	2	0
Tights/pantiehose	0	0	8	2	0
Shoes/boots	0	0	0	4	0
Gloves	0	0	0	0	2

Implement a piece of clothing worn by a non-player character

```
      THE mr_smith ISA ACTOR
        DESCRIPTION
          "blah blah" LIST mr_smith_worn. -- Leave the LIST section out if you
don't want to
          -- have the actor's clothing listed after 'look'.

        VERB examine
          DOES ONLY "blah blah"
            LIST mr_smith. -- This lists what Mr Smith is carrying.
            LIST mr_smith_worn. -- this lists what Mr Smith is wearing.
        END VERB.
      END THE.

      THE mr_smith_worn ISA NPC_WORN -- All containers for clothing worn by NPCs
should be declared ISA NPC_WORN.
         HAS carrier mr_smith. -- The value of the 'carrier' attribute is the
actor wearing the clothes.
      END THE.

      THE bowler_hat ISA CLOTHING IN mr_smith_worn -- now the hat will be described
as being worn by Mr Smith.
         IS headcover 2. -- this attribute must be taken from the clothing table
(above)
      END THE.
```

Remember to declare the piece of clothing NOT takeable if the hero is not supposed to be able to take it!

Implement a lightsource (L)

```
      THE lamp ISA LIGHTSOURCE
        IS NOT natural.
      END THE.

      THE candle ISA LIGHTSOURCE
      END THE.
```

Lightsources are natural or NOT natural. NOT natural lightsources can be turned on and off, lighted and extinguished (= put out). Natural lightsources can only be lighted and extinguished.

Implement a liquid (L)

1)

```
      THE puddle ISA LIQUID AT path
         NAME muddy water NAME puddle
        END THE puddle.
```

2)

```
      THE juice ISA LIQUID IN glass
        HAS vessel glass.
        IS drinkable.
      END THE juice.

      THE glass ISA LISTABLE_CONTAINER AT bedroom
```

```
        END THE glass.
```

An amount of liquid can be taken only if it is in a container.

Note that the container must be stated twice in the code for a liquid:

```
IN glass.
HAS vessel glass.
```

To ensure that all commands behave correctly, liquids are NOT drinkable by default.

Implement a door (L)

```
THE white_door ISA DOOR AT kitchen
  NAME white door
END THE white_door.
```

A door is by default closeable and closed, lockable but not locked. It can be opened and closed if it is not locked.

Implement a door between two rooms so that it opens and closes properly on both sides (L)

In these examples a door is implemented between a bedroom and a corridor:

1)

```
THE bedroom_door1 ISA DOOR AT bedroom
  NAME door
  ...

  VERB open
    DOES MAKE bedroom_door2 open.
  END VERB.
END THE bedroom_door1.

THE bedroom_door2 ISA DOOR AT corridor -- the corridor side of the door
  NAME bedroom door
  ...
  VERB open
    DOES MAKE bedroom_door1 open.
  END VERB.
END THE bedroom_door2.
```

And similarly for the verb 'close'.

You can also locate one and the same door object at the two locations when the hero moves about:

2)

```
THE bedroom_door ISA DOOR AT bedroom
  NAME door
  ...
END THE bedroom_door.
```

```
THE bedroom ISA LOCATION
  ENTERED
  LOCATE bedroom_door AT bedroom.
END THE bedroom.

THE corridor ISA LOCATION
  ENTERED
  LOCATE bedroom_door AT corridor.
END THE corridor.
```

The library takes care of the opening and closing verbs working properly.

3)

```
      EVERY door ISA OBJECT
        HAS otherside door. -- here it is stated that every door has another
side which is also an object
             -- belonging to the door class

        VERB open
          DOES MAKE otherside OF THIS NOT closed. -- opens the other side of
the door as well
        END VERB.

        VERB close
          DOES MAKE otherside OF THIS closed. -- closes the other side of
the door as well
        END VERB.

      END EVERY.
```

And then e.g.:

```
      THE kitchen_door ISA DOOR AT kitchen
        HAS otherside kitchen_door_2. -- the other side of the kitchen door,
e.g. in the living-room
      END THE.

      THE kitchen_door2 ISA DOOR AT livingroom
        HAS otherside kitchen_door.
      END THE.
```

Implement a window (L)

```
THE bedroom_window ISA WINDOW AT bedroom
   NAME bedroom window
END THE bedroom_window.
```

A window is by default closable and closed. It can be opened and closed, looked through and out of.

Implement a supporter (L)

1)

```
THE tray ISA SUPPORTER AT kitchen
END THE tray.

THE coffee_cup ISA OBJECT IN tray -- note the IN
END THE coffee_cup.
```

2)
```
THE table ISA SUPPORTER AT bedroom
END THE table.
```

Implement a supporter that is a container at the same time (e.g. a table with a book on it and two drawers in it) (L)

```
THE table ISA SUPPORTER AT bedroom
  HAS components {drawer1, drawer2}.
  ...

  VERB examine
    DOES
      LIST table.
      FOR EACH c IN components OF THIS
        DO
        SAY "The table has". SAY AN c. "."
          IF c IS NOT closed
            THEN LIST c.
            ELSE SAY THE c. "is closed."
          END IF.
      END FOR.
  END VERB.
...
END THE.

THE drawer1 ISA LISTABLE_CONTAINER
  OPAQUE CONTAINER
  NAME top drawer
  AT bedroom
  IS closed.
END THE.

THE drawer2 ISA LISTABLE_CONTAINER
  NAME bottom drawer
  AT bedroom
  IS NOT closed.
END THE.

THE book ISA OBJECT IN table -- note the IN!
...
END THE book.

THE diary ISA OBJECT IN drawer1
...
END THE diary.
```

In other words, declare the drawer's components of the table, in the manner described above. The result will then be e.g. something like this:

```
      "You see a table here. There is a book on the table. The table has a top
drawer. The top drawer is closed.
      The table has a bottom drawer. The bottom drawer is empty."
```

Implement a readable object (L)

```
THE book ISA OBJECT AT room1
  IS readable.
```

```
     HAS text “blah blah”.
   END THE.
```

Implement an object that can be written in or on (L)

```
   THE notebook ISA OBJECT AT room1
     IS writeable.
   END THE notebook.
```

The player's command should be in the form

```
   >write "text" on blackboard
```

Or

```
    >write "text" in notebook
```

with the text to be written in double quotes.

The player can write text again and again in the same place. The earlier writings are not erased. Thus:

```
   >write "The butler might be the murderer." in notebook

   and, later on,

   >write "But would that be too easy a solution?" in notebook

   will yield

   >read notebook
   The notebook says: "The butler might be the murderer. But would that be too
easy a solution?"
```

Change the attribute of an object

```
   MAKE man hungry.
   MAKE monster attack. -- attributes can be identical to verbs.
   MAKE hero NOT sleepy.
   SET value OF dial TO 2.
   SET bullets_left OF gun TO 1.
```

Bring objects in and out of play

1) To bring an object into play; e.g. in this example a secret trapdoor is revealed when a carpet is moved:

```
   THE livingroom ISA LOCATION
   END THE livingroom.

   THE carpet ISA OBJECT AT livingroom
      VERB look_under
       DOES ONLY
         IF trapdoor NOT AT livingroom
           THEN LOCATE trapdoor AT livingroom.
             "There is a trapdoor under the carpet!"
           ELSE "You see nothing else under the carpet."
```

```
            END IF.
        END VERB.
      END THE carpet.

      THE trapdoor ISA OBJECT -- declare no location for the trapdoor
initially!
      END THE trapdoor.
```

2) To take objects out of play, locate them at a location which isn't used in-game:

```
      THE nowhere ISA LOCATION
      END THE nowhere.

      THE mirror ISA OBJECT AT livingroom
      ...
        VERB break
          DOES ONLY "You break the mirror into thousands of small pieces."
            LOCATE mirror AT nowhere.
        END VERB.
      ...
      END THE mirror.
```

Locate objects somewhere

```
      LOCATE diamond HERE. -- HERE = the current location where the command is
executed
      LOCATE diamond AT treasure_chamber.
      LOCATE diamond AT hero. -- = where the hero is at the moment
      LOCATE diamond IN hero. -- = in the inventory
      LOCATE diamond AT tom. -- = where Tom is at the moment
      LOCATE diamond IN chest.
```

Enable references to immaterial or implied objects

E.g. to whistle a melody, with "melody" being the object referred to:

a)

```
      SYNTAX whistle_melody = whistle melody.
         whistle_melody = whistle 'a' melody.

      VERB whistle_melody
        DOES "You whistle a little melody."
      END VERB.
```

b)

```
      THE melody ISA OBJECT -- declare no location for this object!
        ...
        VERB whistle
          DOES ONLY "You whistle a little melody."
        END VERB.
      END THE melody.

      SYNTAX whistle = whistle (obj)!
        WHERE obj ISA OBJECT
          ELSE "That's not something you can whistle."
```

```
VERB whistle
   DOES "That's not something you can whistle."
END VERB.
```

c)

```
THE melody ISA ENTITY
...
  VERB whistle
    DOES ONLY "You whistle a little melody."
  END VERB.
END THE melody.

SYNTAX whistle = whistle (ent)
  WHERE ent ISA ENTITY
    ELSE "That's not something you can whistle."
```

Entities are present (but invisible) everywhere, so you don't need the ! sign in this last example.

Make distant and out-of-reach objects (L)

```
THE book ISA OBJECT IN shelf
   IS NOT reachable.
END THE.
```

ACTORS (NPC's)

Implement a basic actor

```
THE cat ISA ACTOR AT garden
END THE cat.
```

The default description for an actor is "A cat is here." This description will display after the location description.

Implement an actor with your own description

```
THE cat ISA ACTOR AT garden
  DESCRIPTION "A grey cat is sneaking about in the garden."
END THE cat.
```

This description will display after the location description. You can also do:

```
THE cat ISA ACTOR AT garden
  DESCRIPTION ""
END THE.
```

And then include the description of the cat in the location description itself.

Implement a person (L)

```
THE bob ISA PERSON AT street
```

```
END THE bob.
```

Implement actors that are able to talk (L)

```
THE bob ISA PERSON AT street
END THE bob.
```

`PERSONS` are able to talk (library verbs `ASK`, `TELL`, and `TALK`, among others, will work)

Implement males and females (L)

```
THE bob ISA MALE
END THE bob.

THE helen ISA FEMALE
END THE helen.
```

Males and females are subclasses of 'person', so they have the ability to talk. If you wish to have, e.g. an animal of a specific sex, you should declare:

```
THE cat ISA FEMALE
  CAN NOT talk.
  ...
END THE cat.
```

Make actors act according to scripts

```
THE bob ISA ACTOR AT street
  DESCRIPTION "Bob is standing near you."
  SCRIPT going_to_house.
    STEP "Bob approaches the door of one of the houses along the street."
    STEP "Bob rings the doorbell of the house."
    STEP "The door opens and Bob goes in."
      LOCATE bob AT inside_house.
END THE bob.
```

A script will execute one step for one player turn and stop when all steps are executed.

Start a script for an actor

```
USE SCRIPT going_to_house FOR bob.
```

Change the description of an actor while (s)he is acting out a script

```
THE bob ISA ACTOR AT street
  DESCRIPTION "Bob is not up to anything special at the moment."
  SCRIPT going_to_house.
    DESCRIPTION "Bob is up to something but you are not quite sure what."
    STEP "Bob is here, looking around."
    STEP "Bob rings the doorbell of one of the houses along the street."
    STEP "The door opens and Bob goes in."
      LOCATE bob AT house.
END THE bob.
```

When the script is executing, the script description of Bob will be shown in the location description rather than the normal description of Bob.

Make an actor act out a script over and over

Put `USE` for the current script into its last step.

```
SCRIPT going_to_house.
  DESCRIPTION "Bob is up to something but you are not quite sure what."
    STEP "Bob is here, looking around."
    STEP "Bob rings the doorbell of one of the houses along the street."
    STEP "The door opens and Bob goes in."
      LOCATE bob AT house.
    STEP "The door opens again and Bob comes out."
      LOCATE bob AT street.
      USE SCRIPT going_to_house FOR bob.
```

Here, Bob is continuously going in and out of the house.

Make an actor act out a script only if a condition is fulfilled

1) Here, the hero will be lead to a treasure if he keeps following a non-player character. The character will wait the hero when needed.

```
      THE harry ISA ACTOR AT cave_entrance
      ...
        SCRIPT harry_leads_the_hero_to_the_treasure_vault.
          STEP
            WAIT UNTIL hero HERE. -- The actor won't act out the script
before the hero is at the same location with him.
            "Harry walks deeper into the cave."

            IF harry AT cave_entrance
              THEN LOCATE harry AT cave1.
            ELSIF harry AT cave1
              THEN LOCATE harry AT cave2.
            ELSIF harry AT cave2
              THEN LOCATE harry AT cave3.
            ELSIF harry AT cave3
              THEN LOCATE harry AT treasure_vault.
            END IF.
            IF harry NOT AT treasure_vault
              THEN USE SCRIPT harry_leads_the_hero_to_the_treasure_vault
FOR harry.
            END IF.
      END THE.
```

2) Here, Harry is made to follow the hero. The condition to be fulfilled is then, of course, that the hero is no longer at the same location as Harry.

```
      THE harry ISA ACTOR AT livingroom
      ...
        SCRIPT following_hero.
          STEP WAIT UNTIL hero NOT HERE. -- Harry won't act until the hero
goes to another location.
```

```
            LOCATE harry AT hero. -- Here, Harry is made to follow the hero
immediately.
              "Harry follows you."
            USE SCRIPT following_hero FOR harry.
     END THE.
```

(Following the hero is made automatic in the standard library, see 'Make a non-player character follow the hero' below.)

3) Here, Harry will stop following the hero around if a location is not lit:

```
     THE harry ISA ACTOR AT livingroom
     ...
       SCRIPT following_hero.
         STEP WAIT UNTIL hero NOT HERE.
           LOCATE harry AT hero.
           "Harry follows you."
           IF CURRENT LOCATION IS NOT lit
             THEN "He becomes afraid because of the darkness and stops."
                   STOP harry.
             ELSE USE SCRIPT following_hero FOR harry.
           END IF.
     END THE.
```

`CURRENT LOCATION` refers to the location where an actor acting out a script is located. Usually CURRENT LOCATION is the location of the hero, because in the majority of all cases it is the hero who does the acting, but an active script for a non-player character overrides this.

Make actors go randomly around the game map using scripts

In this example, Bob is going randomly (vertically and horizontally, not diagonally) around in a 3x3 grid. The upper leftmost corner of the grid is L11 and the lower rightmost corner is L33. He is in location L11 to begin with.

```
-- L11 L12 L13
-- L21 L22 L23
-- L31 L32 L33


THE bob ISA ACTOR
  DESCRIPTION ...
  SCRIPT going_around_randomly.
    STEP
      "Bob leaves."
      -- Outputs from events (such as actors moving) will only be printed
if the hero
      -- can "see" them, so the author can output descriptions of all
movements and
      -- actions without being concerned about them being displayed at the
wrong place
      -- or the wrong time.

      IF bob AT L11
        THEN
          IF RANDOM 1 TO 2 = 1
```

```
                THEN LOCATE bob AT L12.
                ELSE LOCATE bob AT L21.
             END IF.
         ELSIF bob AT L12
            THEN
               DEPENDING ON RANDOM 1 TO 3
               = 1 THEN LOCATE bob AT L11.
               = 2 THEN LOCATE bob AT L13
               = 3 THEN LOCATE bob AT L22.
               END DEPEND.
         ELSIF bob AT L13
            THEN
               IF RANDOM 1 TO 2 = 1
                  THEN LOCATE bob AT L12.
                  ELSE LOCATE bob AT L23.
               END IF.
         ELSIF...
            -- (continue in the same vein)
         END IF.
         "Bob enters."
         USE SCRIPT going_around_randomly FOR bob.
END THE bob.
```

Stop an actor from executing a script

```
STOP bob.

STOP cat.
```

Check the number of actors at a given location

```
IF COUNT ISA ACTOR, AT room1 = 5
   THEN "It seems everybody has arrived. The meeting can begin."
END IF.
```

Change the attribute of an actor

```
MAKE hero hungry.

SET tired_level OF bob TO 3.
```

Make actors sitting, standing or lying down (L)

```
1) MAKE hero sitting.

2) MAKE dog lying_down.

3) MAKE hero NOT sitting. -- = the hero stands up

4) THE patient ISA ACTOR IN bed
   IS lying_down.
    END THE patient.
```

An actor is standing by default, if it doesn't have the attributes 'sitting' or 'lying_down'.

Modify the hero character

1) Using the standard library:

Find the code for the hero character at the bottom of the file 'classes.i' and make needed adjustments (add attributes, modify the outcome for the 'examine' command, etc.)

2) Without the standard library:

```
THE hero ISA ACTOR
  HAS health 5.
  VERB examine
    DOES ONLY "How handsome."
  END VERB.
END THE hero.
```

The hero is predefined in the ALAN system but it can be modified through declaring it again, like in (2) above.

Make a non-player character follow the hero (L)

Mid-game:

```
MAKE harry following.
```

From the start of the game:

```
THE servant ISA PERSON
  IS following.
END THE servant.
```

Stop a non-player character from following the hero (L)

```
MAKE harry NOT following.

MAKE servant NOT following.
```

Communicating with actors (L)

The library makes it automatic using the verbs 'ask person about thing', 'ask person for thing' and 'tell person about thing'.

An example of changing the default response:

```
THE ball ISA OBJECT
   ...

  VERB ask
      WHEN topic
      DOES ONLY
        IF act = child
          THEN """It's my ball; don't take it!"", the child says."
        ELSIF act = old_man
          THEN...
        END IF.
   END VERB.
END THE.
```

Commanding actors

It is not possible for the player to insert a comma in the input, in the style of "Tom, take book". Use either a formulation without a comma, or some other kind of verb construction.

a)

```
SYNTAX act_take = (act) take (obj)
  WHERE...
```

Enables player input such as

```
> tom take book
```

b)

```
SYNTAX command_take = tell (act) 'to' take (obj)
  WHERE ...
```

Enables player input such as

```
> tell tom to take book
```

NAMES

Determine the player-referable name of an instance

The word after "THE" will be the name of an instance if nothing else is declared:

```
THE ball ISA OBJECT AT garden
END THE ball.
```

The player will be able to refer to this instance by the name "ball".

However, if you want to use a different name for the instance in your code from the one you will allow the player to use, you should use the NAME construction. This is advisable when the name of the instance is long or when the instance name consists of two or more words:

```
THE bbr ISA LOCATION
   NAME blue bedroom
   DESCRIPTION "..."
   ...
END THE bbr.

THE button1 ISA OBJECT AT room1
   NAME first button
   DESCRIPTION "..."
END THE button1.

THE aunt_betty ISA ACTOR AT garden
   NAME aunt betty
   DESCRIPTION "..."
```

```
END THE aunt_betty.
```

Allow objects to have distinguishing adjectives in front of them

```
THE pillow ISA OBJECT AT bedroom
  NAME small white soft feather pillow
  MENTIONED "white pillow"
  ...
END THE pillow.
```

Here, the player can refer to the pillow with e.g. "small pillow", "white pillow", "soft feather pillow" etc. The rightmost word in a name is understood as the noun of the object. It is required in player input. All other words in front of it are voluntary attributes. This object will be mentioned by the game as "white pillow" (e.g. in the inventory listing and other places) because it was programmed that way after `MENTIONED`. If `MENTIONED` had been left out, the inventory listing of the object would be "small white soft feather pillow".

Allow reference to objects with an adjective only

E.g. instead of the player having to write "push red button", "push blue button" and "push the yellow button", (s)he should be able to just type "push red", "push blue", and "push yellow". Here is how it's allowed:

```
THE red_button ISA OBJECT AT room1
  NAME red button NAME red
END THE red_button.

THE blue_button ISA OBJECT AT room1
  NAME blue button NAME blue
END THE red_button.

THE yellow_button ISA OBJECT AT room1
  NAME yellow button NAME yellow
END THE red_button.
```

Here, 'red', 'blue' and 'yellow' are actually made into nouns from the game's perspective. Note, then, that the same word can function as an adjective or noun for one and the same instance.

Use articles in front of names

The indefinite article for an instance is "a" by default. The definite article is usually "the". They can be printed before object and actor names in-game by using `AN` and `THE`:

```
VERB examine
  DOES "You see nothing special about" SAY THE obj. "."
END VERB.
```

Yields e.g.,

```
>x book
You see nothing special about the book.

VERB take
  CHECK obj IS useful
```

```
      ELSE "You don't need" SAY AN obj. "."
    DOES ...
  END VERB.
```

Yields e.g.,

```
>take candy
You don't need a candy.
```

If you need to change the article (for example, if you need "an"), declare it within the instance:

```
THE owl ISA ACTOR AT woods
  INDEFINITE ARTICLE "an"
END THE.
```

Other alternatives are `DEFINITE ARTICLE` (by default "the"), `NEGATIVE ARTICLE` (by default "any") and or just `ARTICLE`, which is equal to `INDEFINITE ARTICLE`.

If you need "some", e.g. with some uncountable nouns, you have to ways to go:

```
THE money ISA OBJECT IN wallet
  ARTICLE "some"            -- or: INDEFINITE ARTICLE "some"
END THE MONEY.
```

Or

```
THE money ISA OBJECT IN wallet
  NAME some money
  ARTICLE ""
END THE.
```

Display apostrophes in names

```
THE joesbar ISA LOCATION
  NAME Joe''s bar -- put two single apostrophes in a row!
END THE joesbar.
```

Yields

```
Joe's bar
>
```

Allow several alternatives for how to refer to an instance

1)

```
THE note ISA OBJECT IN table
  NAME note NAME paper NAME parchment
  DESCRIPTION "..."
END THE note.
```

2)

```
THE note ISA OBJECT IN table
  DESCRIPTION "..."
END THE note.

SYNONYMS paper, parchment = note.
```

The difference with these two methods is that the first one only involves the specific instance at hand, while in the second case, all possible notes in the g wanted, stick to the first method.

Use reserved words in the ALAN language in instance or location names

```
THE empty_location ISA LOCATION
  NAME 'empty' 'location'
END THE.
```

Both '`empty`' and '`location`' are reserved words in the Alan language. The compiler will misinterpret them if you don't put them inside single quotes in the `NAME` section.

Control how an instance is referred to in-game

Use `MENTIONED`:

```
THE button1 ISA OBJECT AT room1
  NAME big red alarm button
  MENTIONED "red button"
  ...
END THE button1.
```

Yields e.g.

```
"There is nothing special about the red button - but looks may
deceive."
```

If an instance has several `NAME`s but no `MENTIONED` section, the first name is used to refer to the instance in-game:

```
THE note1 ISA OBJECT AT room1
  NAME note NAME parchment NAME paper
END THE note.
```

Yields e.g.

```
"There is nothing special about the note."
```

Note that in a `NAME` statement, single quotes are used (when using reserved words in the ALAN language):

```
THE empty_location ISA LOCATION
  NAME 'empty' 'location'
END THE.
```

However, in a `MENTIONED` statement, double quotes are used.

The principle is that all words to be typed by a player are put inside single quotes in an ALAN code when needed, and all words mentioned by the game itself are placed inside double quotes in an ALAN code.

CLASSES

Make object, actor or location classes

1)
```
EVERY animal ISA ACTOR
END EVERY.

THE cat ISA ANIMAL
END THE cat.

THE lion ISA ANIMAL
END THE lion.

ADD TO EVERY ANIMAL
  VERB stroke
    DOES
      IF THIS = cat
        THEN "She purrs."
        ELSE "That might be dangerous."
      END IF.
END VERB.
END ADD TO.
```
2)
```
EVERY door ISA OBJECT
  IS closable. IS closed. IS lockable. IS NOT locked.

  VERB open
    DOES MAKE THIS NOT closed. -- THIS refers to each instance defined
as belonging to the class 'door'.
  END VERB.

  VERB close
    DOES MAKE THIS closed.
  END VERB.

  VERB knock
    DOES "You knock on" SAY THE THIS. "There is no reply."
  END VERB.
END EVERY.
```

(The verbs above would only apply to all door objects but not other kinds of objects.)

```
THE kitchen_door ISA DOOR AT kitchen
END THE.

THE bathroom_door ISA DOOR AT bedroom
END THE.
```
3)
```
EVERY street ISA LOCATION
...
  VERB cross
    DOES "There's too much traffic; better stay on this side for now."
  END VERB.
END EVERY.

THE 6th_avenue ISA STREET
  NAME 6th avenue NAME street
END THE.

THE broadway ISA STREET
  NAME broadway NAME street
END THE.
```

(A member of a class can have the same name as the class itself.)

Add properties to a class after it has been defined

```
ADD TO EVERY ANIMAL
  IS hungry.
END ADD TO.
```

(gives the same results as defining a new class with the property

```
EVERY animal ISA ACTOR
  IS hungry.
END EVERY.)
```

The `ADD TO` alternative is the more convenient one when using several game files, e.g. extensions.

Check if an instance belongs to a class

1)
```
IF obj ISA SCENERY
  THEN "That's not important."
END IF.
```

2)
```
CHECK obj ISA PERSON
  ELSE "That's not something you can talk to."
```

ATTRIBUTES

Give an attribute to an object, actor or location

An attribute is a word or a literal that describes the characteristics of the instance. The attribute can be anything descriptive of the instance. An attribute is preceded by `IS`, `ARE`, `HAS` or `CAN`; any of these can be used interchangeably to make the code the most readable. `NOT` can additionally be used after these four words.

```
THE monster ISA ACTOR AT cave
  IS sleeping.
  IS NOT hungry.
```

```
...
END THE monster.

THE boy ISA PERSON AT garden
  HAS NOT been_talked_to.
  CAN swim.
...
END THE boy.

THE ball ISA OBJECT AT garden
  HAS play_level 0.
  IS NOT discovered.
...
END THE ball.

THE park ISA LOCATION
  IS open.
  HAS searched_level 0.
...
END THE park.
```

An attribute must not have any spaces in it. If you wish to have two or more words in an attribute, use either an underscore or some other symbol to connect the words together:

```
WRONG:
  HAS NOT been talked to.
  IS NOT extremely busy.
RIGHT:
  HAS NOT been_talked_to.
  IS NOT extremely#busy.
  IS NOT extremelyBusy.
```

Know what attributes are already built-in (L)

There are no built-in attributes in the ALAN language per se. If you are using the standard library, the attributes that are defined there are listed at the top of the file 'verbs.i'. Otherwise there are no limits to conjuring your own attributes:

```
THE wolf ISA ACTOR AT forest
  IS very_hungry.
  IS hunting_for_food.
END THE wolf.
```

Change the attribute of an object, actor or location

```
THE tv ISA OBJECT AT lounge
  IS broken.

  VERB fix
    DOES MAKE tv NOT broken.
  END VERB.

END THE tv.

THE jack ISA ACTOR AT street
  IS NOT friendly.
  CAN NOT sing.

  VERB give
       WHEN act
       DOES
          IF obj = letter
            THEN "Jack accepts it readily."
               MAKE jack friendly.
            ELSE "Jack continues to sulk."
          END IF.
  END VERB.

  VERB teach
     DOES MAKE jack sing.
  END VERB.
END THE jack.

THE bedroom ISA LOCATION
  HAS value 1.
  IS NOT cleaned.

  VERB clean
     DOES "You clean the bedroom."
       MAKE bedroom cleaned.
  END VERB.

  VERB jump
     DOES INCREASE value OF bedroom.
  END VERB.

  VERB sing
     DOES DECREASE value OF bedroom.
  END VERB.

END THE bedroom.
```

Make an instance behave differently when an attribute changes

```
THE tv ISA OBJECT AT lounge
  IS broken.

  VERB watch
     DOES
       IF tv IS broken
          THEN "You can't; the tv is broken."
          ELSE "You watch the tv."
       END IF.
  END VERB.

END THE tv.
```

Increase a numeric attribute

```
THE hero ISA ACTOR
   HAS strength 10.
END THE hero.

THE magic_potion ISA OBJECT AT cottage
```

```
  VERB drink
    DOES INCREASE strength OF hero.
  END VERB.
END THE magic_potion.

By default, INCREASE increases the numeric attribute by 1.
To have it some other way, use BY:

INCREASE strength OF hero BY 3.
INCREASE difficulty OF game BY 5.
```

Decrease a numeric attribute

```
THE bottle ISA OBJECT AT basement
  HAS level 5.

  VERB drink
    DOES ONLY
      IF level OF bottle >= 1
        THEN
          "You take a sip from the bottle."
          DECREASE level OF bottle.
      END IF.
      IF level OF bottle = 0
        THEN "It is now empty."
      END IF.
  END VERB.
END THE bottle.
```

By default, `DECREASE` reduces the numeric attribute by 1. To have it some other way, use `BY`:

```
DECREASE level OF bottle BY 2.
DECREASE power OF monster BY 5.
```

Check the value of a numeric attribute

1)

```
IF level OF bottle = 0
  THEN "The bottle is empty!"
END IF.
```

2)

```
WHEN tiredness OF hero > 5
  THEN MAKE hero asleep.
    "You fall asleep on the spot."
```

3)

```
VERB jump
   DOES
    IF strength OF hero >= 5
      THEN "You jump easily over the fence."
    END IF.
END VERB.
```

4)

```
DEPENDING On weight Of obj
```

```
    = 1 THEN "light as a feather"
    BETWEEN 2 AND 10 THEN "carryable"
    BETWEEN 11 AND 20 THEN "heavy"
    > 20 THEN "immobile"
    ELSE "weightless"
  END DEPEND.
```

Use random values

1)

```
SET value OF dice TO RANDOM 1 TO 6.
```

2)

```
DEPENDING ON RANDOM 1 TO 3
= 1 THEN "This is the first alternative message."
= 2 THEN "This is the second alternative message."
= 3 THEN "This is the third alternative message."
END DEPEND.
```

3)

```
IF RANDOM 1 TO 2 = 1
  THEN "Yippee!"
  ELSE "Oh well."
END IF.
```

4) Random values cannot be used in attributes:

WRONG:

```
THE hero ISA ACTOR
  HAS health RANDOM 3 TO 6.
END THE hero.
```

Use `INITIALIZE` instead:

RIGHT:

```
THE hero ISA ACTOR
  HAS health 0.
  INITIALIZE
    SET health OF hero TO RANDOM 3 TO 6.
END THE hero.
```

SETS

Use sets in general

```
THE cathy ISA PERSON
  HAS friends {tim, tom, helen, tammy}.
  DESCRIPTION ...
...
END THE cathy.
```

The members of a set must be listed inside curly brackets {}. The members of a set must be instances declared elsewhere in the code. A set must have at least one member. Each member can only occur once in the same set, but a member can occur in multiple sets.

Include something in a set

```
INCLUDE greg IN friends OF cathy.
```

Exclude something from a set

```
EXCLUDE greg FROM friends OF cathy.
```

To check that something is in a certain set

```
IF greg (NOT) IN friends OF cathy
  THEN...
END IF.
```

Make something to happen only to the members of a set

```
VERB test
  DOES
    FOR EACH act ISA ACTOR, IN friends OF cathy
      DO
      MAKE act invited_to_party.
    END FOR.
END VERB.
```

Make something happen to a random member of a set

Here, the game chooses a random member of a group of suspects and makes him/her the criminal.

```
THE detective ISA ACTOR AT livingroom
  HAS suspects {abe, bill, mary, tom, vivian}.
  INITIALIZE
    MAKE RANDOM IN suspects OF detective guilty.
END THE detective.
```

OPERATORS

Use operators in general

- The relational operators are: `=`, `<`, `>`, `<=`, `>=` and `<>`.
- The binary operators are: `+`, `-`, `*` and `/`.
- A special operator is `BETWEEN`.

They are used to check the value of an attribute or to set the value of an attribute.

Examples of use:

1)
```
IF level OF bottle > 5
  THEN ...
  ELSE ...
```

```
        END IF.
```
2)
```
        VERB inventory
          DOES
          "You have" SAY COUNT ISA treasure, IN hero. "treasures presently."
          IF COUNT ISA treasure, IN hero < 10
            THEN "You need" SAY 10 - COUNT ISA treasure, IN hero "more
treasures to
            enter the temple."
          END IF.
        END VERB.
```
3)
```
        SAY amount OF dollars + 50.
```
4)
```
        SET health OF hero TO health OF hero + 3.
```

(This is equal to:

```
        INCREASE health OF hero BY 3.)
```
5)
```
        IF value OF car * 2 < value OF house
          THEN...
        END IF.
```
6)
```
        CHECK value OF house BETWEEN 50000 and wealth OF hero
          ELSE...
```
7)

```
        CHECK recipient <> hero
          ELSE "You can't give something to yourself!"
```

VERBS

Implement new verbs

1) Verbs with no objects, e.g. >dream

```
        SYNTAX dream = dream.

        VERB dream
          DOES "You dream."
        END VERB.
```

2) Verbs with one object, e.g. >paint wall, >find out about mr who

```
        SYNTAX paint = paint (obj)
          WHERE obj ISA OBJECT
            ELSE "That's not something you can paint."

        ADD TO EVERY OBJECT
        VERB paint
          DOES "You paint" SAY THE obj. "."
          -- (MAKE obj painted.)
          -- (DECREASE amount_left OF red_paint BY 2.)
        END VERB.
```

```
END ADD TO.
```

In addition to the verb having been restricted to only apply to objects in the syntax, the `ADD TO` construction is needed in connection with the verb definition, as well:

```
SYNTAX find_out = find 'out' about (act)
   WHERE act ISA ACTOR
     ELSE "That's not a thing you can find something out about."

ADD TO EVERY ACTOR
  VERB find_out
    DOES "You can't find out anything new about" SAY THE act. "."
  END VERB.
END ADD TO.
```

3) Verbs with two objects, e.g. >point wand at monster

```
SYNTAX point = point (obj) 'at' (target)
  WHERE obj ISA OBJECT
    ELSE "That's not something you can point."
  AND target ISA THING
    ELSE "You can't point anything at that."
```

-- `THING` refers to both objects and actors

```
ADD TO EVERY THING
VERB point
  DOES "You point" SAY THE obj. "at" SAY THE target. "."
END VERB.
END ADD TO.
```

In these examples, the parameters (`act`, `obj`, `target`) could have been named in any other way; there is no reason (except for the sake of easily readable code) that they were named this way above. You could just as well have e.g.

```
SYNTAX point = point (ghg) 'at' (df22)
  WHERE ghg ISA OBJECT
    ELSE "That's not something you can point."
  AND df22 ISA THING
    ELSE "You can't point anything at that."

ADD TO EVERY THING
VERB point
  DOES "You point" SAY THE ghg. "at" SAY THE df22. "."
END VERB.
END ADD TO.
```

However, `OBJECT`, `ACTOR` and `THING` must be exactly stated the way they are in the examples.

Use multiple syntaxes for the same verb

```
SYNTAX give = give (obj) to (act) -- e.g. "give the ball to the boy"
  WHERE obj ISA OBJECT
    ELSE...
  AND act ISA ACTOR
```

```
    ELSE...

  give = give (act) (obj). -- e.g. "give the boy the ball"


SYNTAX take = take (obj)
  WHERE obj ISA OBJECT
    ELSE...

  take = pick 'up' (obj).

  take = pick (obj) 'up'.
```

The parameters need to be explained only in the first syntax; the same parameters used in other syntax formulations don't require explanations.

Make two or more verbs have similar outcomes

```
EVERY SCENERY ISA OBJECT
  VERB examine, take
    DOES SAY THE obj. “is not interesting.”
  END VERB.
END EVERY.
```

In order for this to work, the verbs in these constructs should have similar syntaxes (That is to say, the number of parameters should be equal and they should be named in the same way).

Create synonyms for verbs

```
VERB take
  DOES LOCATE obj IN hero.
    "Taken."
END VERB.

SYNONYMS get, grab, confiscate = take.
```

It is not possible for a synonym to have two elements. E.g. 'pick up' would not be acceptable in the above example. You would have to make 'pick up' an alternative syntax for the 'take' verb (see 'Use multiple syntaxes for the same verb' above.)

Use default syntaxes to ease programming

If you declare a verb inside an instance, the verb will automatically have the syntax 'verb (instance)'. The verb will then work only with that instance and no other ones. E.g. this coding enables the player to type 'cross street' and get a response:

```
THE street ISA OBJECT AT town
...
  VERB cross
    DOES "There's too much traffic."
  END VERB.
END THE street.
```

Note that this doesn't apply to locations. If you declare:

```
THE basement ISA LOCATION
  DESCRIPTION ...

  VERB take
    DOES ONLY "There's nothing worth taking among the old junk here."
  END VERB.
END THE.
```

The 'take' verb applies to objects in the basement, not to the basement itself.

If a verb is declared without a syntax and outside any instances, it will have the syntax 'verb'.

```
VERB test
  DOES "Test successful."
END VERB.
```

Here, if the player typed 'test' in-game, (s)he would get the response "Test successful."

Program ditransitive verbs

Sometimes an instance might be either one of two parameters of a ditransitive verb. Then the two different cases need to be singled out separately:

```
      SYNTAX ask_about = ask (act) about (topic)
        WHERE act ISA ACTOR
          ELSE ...
        AND topic ISA THING
          ELSE ...

      THE man ISA ACTOR AT street
      ...
        VERB ask
            WHEN act -- when the 'man' is the one asked
            DOES ONLY "The man doesn't seem to know anything about" SAY THE
topic. "."
              WHEN topic -- when the 'man' is the topic asked about
            DOES ONLY SAY THE act. "doesn't seem to know much about the man."
        END VERB.
      END THE man.
```

Enable verbs to refer to instances that are not present

Use an exclamation mark after the instance in the syntax.

```
SYNTAX think_about = think about (obj)!
  WHERE obj ISA THING
    ELSE "That's not something you can think about."
```

The inclusion of the exclamation mark ensures that the verb will then take into account objects in other locations, as well, and not just those at the present location of the hero.

Enable verbs to refer to multiple objects, or to "all"

Use an asterisk after the instance in the syntax.

```
SYNTAX take = take (obj)*
   WHERE obj ISA OBJECT
    ELSE "That's not something you can take."
```

This allows e.g.:

```
>take the vase and the plate
(vase) Taken.
(plate) Taken.

>take all
(vase) Taken.
(plate) Taken.
```

If you leave the asterisk out, the outcome will be:

```
"You can't refer to multiple objects with 'take'."
```

Make verbs act differently in a given location

```
      THE basement ISA LOCATION
        DESCRIPTION ...

        VERB jump
          DOES ONLY "The ceiling is too low here."
        END VERB.

        VERB take
          DOES ONLY "There is nothing worth taking here; everything around you is
old junk."
        END VERB.

      END THE basement.
```

Making verbs act differently in a given situation

```
VERB jump
   DOES
    IF hero IS sleepy
      THEN "You're too sleepy for that."
      ELSE "You make a jump."
    END IF.
END VERB.
```

Change the default outcomes for actions as they are defined e.g. in the standard library (L)

You should open the library file (e.g. 'verbs.i') and change the default outcome message for the action (i.e., the message after `DOES`).

Here is an example for 'jump':

First, locate this verb in the file 'verbs.i'. Its definition looks like this:

```
 ----------------------
VERB jump
  CHECK hero IS NOT sitting
    ELSE "It is difficult to jump while sitting down."
  AND hero IS NOT lying_down
    ELSE "It is difficult to jump while lying down."
  DOES
  "You jump on the spot, to no avail."
END VERB.
----------------------
```

Then, change the line "You jump on the spot, to no avail." to your own liking, e.g. "You make a jump but nothing happens."

```
----------------------
VERB jump
  CHECK hero IS NOT sitting
    ELSE "It is difficult to jump while sitting down."
  AND hero IS NOT lying_down
    ELSE "It is difficult to jump while lying down."
  DOES
  "You make a jump but nothing happens."
END VERB.
-----------------------
```

And that's it!

Change the default outcome of a command for a specific object So that it differs from the one defined e.g. in the standard library (L)

Use `DOES ONLY`:

```
THE ball ISA OBJECT AT garden
...
  VERB kick
    DOES ONLY "You kick the ball. It lands on the other side of the fence."
      LOCATE ball AT greener_lawn.
  END VERB.
END THE ball.
```

Restrict verbs so that all or a number of actions yield the same message in a given situation

Use the extension "Restricted verbs" available on the ALAN website.

Make checks for verbs

```
VERB sing
  CHECK aunt_mary NOT AT hero
```

```
          ELSE "You know very well that aunt Mary doesn't approve of your
singing."
        AND throat IS NOT sore
          ELSE "You still haven't taken the throat pills."
        AND lyrics ARE found
          ELSE "You still haven't found the sheet music."
        DOES "You sing."
      END VERB.
```

If a verb has a `CHECK` only and no `ELSE` or `DOES`, the action will stop in all cases:

```
      THE ball ISA OBJECT AT garden
        VERB kick
          CHECK "You're not interested."
        END VERB.
      END THE.
```

In practice, this is similar to using `DOES ONLY`.

Check where an instance is

```
      CHECK diamond (NOT) AT treasure_chamber
        ELSE ...
      AND spellbook IN hero -- in the hero's inventory
        ELSE ...
      AND servant AT hero -- in the same location as the hero
        ELSE ...
      AND princess NEARBY -- any adjacent location connected by an exit to the
current location
        ELSE ...
      AND horse NEAR main_gate -- any adjacent location to the one mentioned, and
connected by an exit to it
        ELSE ...
```

Check an attribute of the current location

```
      VERB read
        CHECK CURRENT LOCATION IS lit
          ELSE "It's too dark to see!"
        DOES "You read."
      END VERB.
```

Add or edit checks for verbs in the standard library (L)

Most verbs and commands are in the file 'verbs.i'. Scroll down the alphabetical list to the verb you wish to edit. Look if the existing checks are enough, or add one to a convenient place in the code. For example, here is an example with the verb 'jump' where we add a check that verifies that the hero is fit enough to make the jump. Our additional check is here placed after the two default checks in the library:

```
      VERB jump
        CHECK hero IS NOT sitting
          ELSE "It is difficult to jump while sitting down."
        AND hero IS NOT lying_down
           ELSE "It is difficult to jump while lying down."
```

```
    AND strength OF hero > 3
      ELSE "You don't feel strong enough."
    DOES
      "You jump on the spot, to no avail."
  END VERB.
```

In the following example, the two default checks for the verb 'jump' are modified:

```
  VERB jump
     CHECK hero IS NOT sitting
      ELSE "How about getting off the chair first?"
    AND hero IS NOT lying_down
      ELSE "Making a jump while lying down is a difficult feat to accomplish."
    DOES
      "You jump on the spot, to no avail."
  END VERB.
```

If you wish to have a check apply to one instance only, place the check in the verb under that instance:

```
  THE soup ISA OBJECT AT kitchen
    IS NOT hot.

    VERB eat            -- declared in the library
      CHECK soup IS hot
        ELSE "You'll have to heat the soup before eating it."
    END VERB.

  END THE.
```

Notice that there is no `DOES` section above. If the above check is passed, the `DOES` section of the 'eat' verb, as it is declared in the standard library, will be carried out.

Make a verb yield varied outcome messages

```
  VERB take
    CHECK ...
  DOES
    DEPENDING ON RANDOM 1 TO 3
      = 1 THEN "Taken."
      = 2 THEN "You take" SAY THE obj. "."
      = 3 THEN "You pick up" SAY THE obj. "."
    END DEPEND.
  END VERB.
```

Make something happen before the default outcome of a verb, e.g. as stated in the standard library

E.g.

```
  >read parchment
  (first unfolding the parchment)
  You read the parchment.
```

Use `DOES BEFORE`:

```
THE parchment ISA OBJECT
   IS readable.
  ...
  VERB read
    DOES BEFORE "(first unfolding the parchment)"
  END VERB.
...
END THE parchment.
```

Make something happen after the default outcome of a verb, e.g. as stated in the library

E.g.

```
>read parchment
 You read the parchment.
 After reading it you fold it again and put it back to where you found it.
```

Use `DOES AFTER`:

```
THE parchment ISA OBJECT
   IS readable.
  ...
  VERB read
    DOES AFTER "After reading it you fold it again and put it back to where
you found it."
  END VERB.
  ...
END THE parchment.
```

Implement implicit taking (L)

E.g.

```
>eat apple
 (first taking the apple)
 You eat the apple. It tastes delicious.
```

This is handled automatically by the standard library.

Override automatic implicit taking (L)

You have to delete the implicit taking code manually for each verb that you wish to delete it for. The implicit taking code typically looks like this:

```
-- implicit taking:
IF obj NOT DIRECTLY IN hero
  THEN "(taking" SAY THE obj. "first)$n"
  LOCATE obj IN hero.
END IF.
-- end of implicit taking.
```

Delete this from the `DOES` section of the verb(s) in question. Moreover, you'll have to add an extra check to the verb:

```
AND obj NOT IN hero
  ELSE "You don't have" SAY THE obj. "."
```

Thus, e.g. the verb 'eat', from 'verbs.i', would look like this, with the implicit taking removed:

```
ADD TO EVERY OBJECT
  VERB eat
    CHECK food IS edible
      ELSE
        IF food IS NOT plural
          THEN "That's not"
          ELSE "Those are not"
      END IF.
        "something you can eat."
    AND food IS takeable
      ELSE "You don't have" SAY THE food. "."
    AND CURRENT LOCATION IS lit
      ELSE "It is too dark to see."
    AND food IS reachable
      ELSE SAY THE food.
        IF food IS NOT plural
          THEN "is"
          ELSE "are"
        END IF.
        "out of your reach."
    AND obj NOT IN hero
       ELSE "You don't have" SAY THE obj. "."
    DOES
      "You eat all of" SAY THE food. "."
      LOCATE food AT nowhere.
  END VERB.
 END ADD.
```

CONDITIONAL STATEMENTS

Use IF-statements

IF-statements cannot be used independently. They are found only in verb definitions, checks, descriptions etc.

a)

```
VERB jump
  DOES
    IF hero AT basement
      THEN "The ceiling is too low here."
      ELSE "You make a jump."
    END IF.
END VERB.
```

b)

```
THE house ISA OBJECT AT garden
  DESCRIPTION
    IF house IS NOT 'entered'
      THEN "..."
      ELSE "..."
```

```
            END IF.
        END THE house.
```

c)

```
        VERB attack
          CHECK monster IS asleep
            ELSE
              IF monster IS hungry
                THEN...
                ELSE...
              END IF.
            DOES ...
        END VERB.
```

The logic operators `AND` and `OR` can be used in `IF`-statements:

```
IF mr_burton AT office OR mr_burton AT street
   THEN...
END IF.

IF floor IS vacuumed AND dishes ARE washed AND livingroom IS cleaned
  THEN...
END IF.
```

Parentheses can be used to affect the order of execution:

```
IF (mr_burton AT office OR mr_smith AT office) AND ms_stanton AT office
  THEN...
END IF.
```

EVENTS

Use events in general

An event must be triggered by `SCHEDULE` outside the event code itself.

If you want the event to repeat over several turns, you must put `SCHEDULE` inside the event code, too.

Events are cancelled using `CANCEL`.

The code for an event stands independently, outside any instance or verb definitions.

Implement a single one-time event (e.g. an explosion)

In this example, an explosion takes place five turns after the game start:

```
THE street ISA LOCATION
END THE street.

EVENT explosion
  "$pYou hear a big bang somewhere nearby."
END EVENT.

START AT street.
```

```
    SCHEDULE explosion AT hero AFTER 5.
```

The "`AT hero`" ensures that wherever the hero is at the fifth turn, he will hear the explosion. If you said "`SCHEDULE` explosion AT street `AFTER 5`", the hero wouldn't hear the explosion if he wasn't in the street at that moment.

Implement a one-time event that takes place right at the beginning of the game:

```
THE street ISA LOCATION
END THE street.

EVENT explosion
   "$pYou hear a big bang somewhere nearby."
END EVENT.

START AT street.
  SCHEDULE explosion AT street AFTER 0.
```

Make an event be triggered by an action

```
VERB jump
  CHECK hero AT kitchen
    ELSE "Nothing happens."
  AND floor NOT broken
    ELSE "You've wrecked the floor here already."
  DOES SCHEDULE floor_collapse AT room1 AFTER 0.
END VERB.

EVENT floor_collapse
  "The floor collapses under you! You fall into the basement."
  LOCATE hero AT basement.
  MAKE floor broken.
  MAKE basement lit.
END EVENT.
```

Implement repeated events with the same message displaying every time (e.g. "The phone keeps ringing.")

```
EVENT phone_ringing
  "$pThe phone keeps ringing."
  SCHEDULE phone_ringing AT office AFTER 1.
END EVENT.

START AT office.
  "You were having a pleasant afternoon nap in your office when you were
rudely woken up by
  the irritating ringing of your work phone. Bemused, you rub your eyes and
sit up in your chair."
    SCHEDULE phone_ringing AT office AFTER 1.
```

Implement repeated events with varying messages

1) after the messages have each appeared once, the event stops

2) after the messages have each appeared once, the whole cycle is repeated over and over again

3) messages are displayed in random order, and after all of them have been displayed at least once, the event stops

4) messages are displayed in random order for a number of turns, and even if all of them haven't necessarily been displayed yet, the event stops

5) messages are displayed in random order over and over again

As follows:

1) after the messages have each appeared once, the event stops

```
     EVENT walls_approach
       IF value OF walls = 0
         THEN "The walls of the room start moving, approaching you!"
       ELSIF value OF walls = 1
         THEN "The walls keep approaching you! The free space in the room gets
smaller and smaller."
       ELSIF value OF walls = 2
         THEN "There is only a narrow passage left in the middle of the room. You
had better do something quickly!"
       ELSIF value OF walls = 3
         THEN "The walls crush you!"
           CANCEL walls_approach. -- this is not necessarily needed because of
the QUIT following below
           QUIT.
       END IF.
       INCREASE value OF walls.
       SCHEDULE walls_approach AT hero AFTER 1.
     END EVENT.

     THE walls ISA OBJECT AT room1
       HAS value 0.
       ...
       VERB touch
         DOES
           IF value OF walls = 0
             THEN SCHEDULE walls_approach AT hero AFTER 0.
             ELSE "Nothing special happens."
           END IF.
       END VERB.
     END THE walls.


     (To make the walls stop, just do e.g.

     VERB simsalabim
       DOES "That helped! The walls retreat to their original position. You are
safe."
         CANCEL walls_approach.
     END VERB.)
```

2) after the messages have each appeared once, the whole cycle is repeated over and over again

```
     EVENT streetlife
       IF value OF street = 1
```

```
          THEN "A neverending line of cars passes by."
        ELSIF value OF street = 2
          THEN "Busy people going to work rush past you."
        ELSIF value OF street = 3
          THEN "A street vendor is striving to sell hotdogs to the busy passers-
by."
        ELSIF value OF street = 4
          THEN "The noise of the traffic together with the almost unbearable heat
really starts getting on your nerves."
            SET value OF street TO 0.
        END IF.
        INCREASE value OF street.
        SCHEDULE streetlife AT hero AFTER 1.
      END EVENT.

      THE street ISA OBJECT AT town
        HAS value 1.
      END THE street.

      START AT town.
         SCHEDULE streetlife AT town AFTER 0.
```

3) a number of messages is displayed in random order, and after all of them have been displayed at least once, the event stops

Here, all five atmopsheric messages in a forest must display at least once before the event stops.

```
      EVENT forest_messages
        DEPENDING ON RANDOM 1 TO 5
        = 1 THEN "In the distance you hear the chirping of a songbird."
          INCREASE 1_message_value OF forest.
        = 2 THEN "A small fox peeks from under one of the bushes and disappears a
moment later."
          INCREASE 2_message_value OF forest.
        = 3 THEN "You feel a light, weclome breeze on your face."
          INCREASE 3_message_value OF forest.
        = 4 THEN "A white butterfly flutters past."
          INCREASE 4_message_value OF forest.
        = 5 THEN "A wasp circles you for a couple of times and flies away."
          INCREASE 5_message_value OF forest.
        END DEPEND.
        SCHEDULE forest_messages AT hero AFTER 1.
      END EVENT.

      THE forest ISA LOCATION
        HAS 1_message_value 0.
        HAS 2_message_value 0.
        HAS 3_message_value 0.
        HAS 4_message_value 0.
        HAS 5_message_value 0.
        ENTERED
          SCHEDULE forest_messages AT hero AT forest AFTER 0.
        ...
      END THE forest.

      WHEN 1_message_value OF forest >= 1 AND 2_message_value OF forest >= 1 AND
3_message_value OF forest >= 1
        AND 4_message_value OF forest >= 1 AND 5_message_value OF forest >= 1
      THEN CANCEL forest_messages.
```

4) messages are displayed in random order for a number of turns, and even if all of them haven't necessarily been displayed yet, the event stops

In this example, five different messages are displayed randomly for 12 turns.

```
 EVENT forest_messages
   DEPENDING ON RANDOM 1 TO 5
   = 1 THEN "In the distance you hear the chirping of a songbird."
   = 2 THEN "A small fox peeks from under one of the bushes and disappears a
moment later."
   = 3 THEN "You feel a light, welcome breeze on your face."
   = 4 THEN "A white butterfly flutters past."
   = 5 THEN "A wasp circles you for a couple of times and flies away."
   END DEPEND.
   SCHEDULE forest_messages AT hero AFTER 1.
   INCREASE value OF forest_message_counter.
   IF value OF forest_messages_counter > 12
     THEN CANCEL forest_messages.
       "You feel tired of observing your surroundings and concentrate on
finding the way out of the forest."
   END IF.
END EVENT.

THE forest_messages_counter ISA OBJECT
  HAS value 1.
END THE.

THE forest ISA LOCATION
  ENTERED
    SCHEDULE forest_messages AT hero AT forest AFTER 0.
...
END THE forest.
```

5) messages are displayed in random order over and over again

```
EVENT forest_messages
  DEPENDING ON RANDOM 1 TO 5
  = 1 THEN "In the distance you hear the chirping of a songbird."
  = 2 THEN "A small fox peeks from under one of the bushes and disappears a
moment later."
  = 3 THEN "You feel a light, welcome breeze on your face."
  = 4 THEN "A white butterfly flutters past."
  = 5 THEN "A wasp circles you for a couple of times and flies away."
   END DEPEND.
  SCHEDULE forest_messages AT hero AFTER 1.
END EVENT.
```

Control the frequency how often the messages in a repeated event are displayed

(e.g. every turn, every three turns,every 2 to 5 turns, or waiting for a certain happening before the next message in an event is displayed)

1) Every turn:

Schedule the event to happen again on the next turn like this:

```
EVENT event1
  "This message is displayed every turn."
  SCHEDULE event1 AT room1 AFTER 1.
END EVENT.
```

2) Every three turns:

```
EVENT event1
   "This message is displayed every three turns."
  SCHEDULE event1 AT room1 AFTER 3.
END EVENT.
```

3) Every 2 to 5 turns:

```
EVENT event1
  "This message is displayed every two to five turns."
  SCHEDULE event1 AT room1 AFTER RANDOM 2 TO 5.
END EVENT.
```

4) waiting for something to happen before the next message in an event is displayed:

In this example, an event occurs only if a button is pushed:

```
EVENT what_the_heck
  IF value OF button = 1
    THEN "A pink ghost emerges from the closet!"
  ELSIF value OF button = 2
    THEN "A bowl of ice cream appears on the table!"
  ELSIF value OF button = 3
    THEN "A ominous, maniacal laughter fills the room."
  ELSIF value OF button = 4
    THEN "The furniture starts floating!"
  ELSIF value OF button = 5
    THEN "The button itself vanishes!"
    LOCATE button AT nowhere.
  END IF.
  INCREASE value OF button.
END EVENT.

THE button ISA OBJECT AT room1
  HAS value 1.

  VERB push
    DOES ONLY
      SCHEDULE what_the_heck AT room1 AFTER 0.
  END VERB.

END THE button.
```

Stop a repeated event or cancel an event before it has happened

```
CANCEL phone_ringing.

CANCEL explosion.
```

Determine whether an event has already happened

Events cannot have attributes (e.g. 'happened' or 'NOT happened'.)

Make a control object that changes its value when the event executes and check its value when needed:

```
EVENT e1
  "You approach Bob in order to pop the question but he gives the appearance
of
  not wanting to cooperate in the least. You decide it's better to leave him
be."
  INCREASE value OF e1_object.
END EVENT.

THE e1_object ISA OBJECT
  HAS value 0.
END THE e1_object.

THE bob ISA OBJECT
  VERB ask
      WHEN person
      CHECK value OF e1_object = 0
      ELSE "It seems there is no help to be expected from Bob."
    DOES SCHEDULE e1 AT hero AFTER 0.
  END VERB.
END THE bob.
```

Determine how many times a certain event has executed

The following example implements a secret passage that opens when a piano is played, but only for the first two tries.

```
EVENT e1
  "You press a few keys of the piano. A secret door opens in the wall!"
    MAKE secret_door NOT closed.
    INCREASE value OF e1_object.
END EVENT.

THE e1_object ISA OBJECT
  HAS value 0.
END THE e1_object.

THE piano ISA OBJECT AT livingroom
  VERB 'play'
    DOES ONLY
      IF value OF e1_object > 2
        THEN "The secret door won't open anymore. I guess the trick only
worked for a couple of times."
        ELSE SCHEDULE e1 AT hero AFTER 0.
      END IF.
  END VERB.
END THE piano.
```

LOOPS

Use FOR EACH formulations

You can affect a group of instances simultaneously by using FOR EACH. The affected group can be a class, or instances sharing an attribute or a location, or the members of a set.

1)
```
FOR EACH a1 ISA ANIMAL
  DO
  MAKE a1 sleepy.
END FOR EACH.        -- or: END FOR. or: END EACH.
```

2)
```
FOR EACH p ISA PERSON, IN livingroom
  DO
  MAKE p suspect OF detective.
END EACH.
```

3)
```
FOR EACH o ISA OBJECT, IS NOT reachable, IN storage
  DO
  MAKE o reachable.
END FOR.
```

A temporary variable is needed in the `FOR EACH` formulation. (In the above examples these variables are a1, p and o. The variables can be named in any way the game author chooses to name them.)

RULES

Use rules in general

The code for a rule must stand independently, outside any instance or verb definitions:

1)
```
      WHEN speed OF car = 240
        THEN MAKE car overheated.
```
2)
```
      WHEN hero AT home AND dishes ARE washed AND livingroom_floor IS
vacuumed
        THEN MAKE hero sleepy.
```

3)
```
      WHEN dishes ARE washed OR livingroom_floor IS vacuumed
        THEN MAKE doorbell ringing.
```
4)
```
      WHEN (mr_benson AT garden OR mrs_benson AT garden) AND postman AT
street
        THEN SCHEDULE conversation AT garden AFTER 0.
```
5)
```
      WHEN hero AT forest_clearing AND bear IS hungry
        THEN SCHEDULE bear_approaching_message AT hero AFTER 0.

      EVENT bear_approaching_message
        "$pA hungry bear approaches you!"
      END EVENT.
```

Rules cannot trigger messages in-game, they just alter game state to have messages triggered after rules, use `SCHEDULE` and put the message into an `EVENT`.

SCRIPTS

See -> Actors

STRINGS

Use strings in commands (e.g. 'answer "ball"')

```
SYNTAX answer = answer (str)
  WHERE str ISA STRING
    ELSE "That's not something you can answer."

VERB answer
   DOES
    IF str = "ball"
      THEN "That's correct! Congrats!"
      ELSE "Sorry, that was wrong."
    END IF.
END VERB.
```

Check if there is a certain word or character in a string

```
IF str CONTAINS "why"
  THEN "Because I say so."
END IF.

IF str NOT CONTAINS "10"
  THEN "Sorry, that was wrong."
END IF.
```

TEXT FORMATTING

Attach text right after previous text

Nothing special needs to be taken into account. Sentences follow each other naturally. There is no need to leave any extra spaces anywhere; the next sentence continues so that a space is automatically left:

```
THE door ISA OBJECT AT room1
  IS closable. closed.

  VERB examine
    DOES ONLY "It's a white door."
      IF door IS closed
```

```
            THEN "It is closed."
            ELSE "It is open."
          END IF.
      END VERB.
    END THE door.
```

Yields:

```
    >x door
    It's a white door. It is closed.
```

There will be no space before a full stop or a comma:

```
    THE door ISA OBJECT AT room1
      IS closable. closed.

      VERB examine
        DOES ONLY "It's a white door which is"
          IF door IS closed
            THEN "closed"
            ELSE "open"
          END IF.
          "."
      END VERB.

    END THE door.

    THE door ISA OBJECT AT room1
       IS closeable. closed.

      VERB examine
        DOES ONLY "A white door"
          IF door IS closed
            THEN ", currently closed,"
            ELSE ", currently open,"
          END IF.
          "leads into room2."
      END VERB.

    END THE door.
```

If you need no space in some cases, use $$:

```
    EVERY DOCUMENT ISA OBJECT
      MENTIONED “document”
        IF COUNT, ISA DOCUMENT, IN folder > 1
          THEN “$$s”
        END IF.
    END EVERY.
```

Yields e.g.:

```
    > x folder
    In the folder you see some documents.
```

Make a body of text start at a new line

Insert "`$n`" before the the text you wish to start at a new line:

```
      THE ball ISA OBJECT
        VERB examine
          DOES "It is an ordinary basketball.$nRound.$nSmooth.$nOrange."
        END VERB.
      END THE.
```

Yields:

```
      >x ball
      It is an ordinary basketball.
      Round.
      Smooth.
      Orange.
```

Start a new paragraph (I.e. make the text continue after one empty line)

Insert "$p" before the text you wish to continue after one empty line:

```
      THE outside_house ISA LOCATION
        DESCRIPTION "A white wooden house stands on the top of a hill.
          $pChildren are playing in the garden."
      END THE outside_house.
```

Yields

```
      >look
      House
      A white wooden house stands on the top of a hill.

      Children are playing in the garden.
```

Indent the text on a new line

Insert "$i" before the intended indentation:

```
      START AT garden.
      "You thought today would be just another ordinary day.$iYou couldn't have
been more wrong."
```

Yields:

```
      You thought that today would be just another ordinary day.
        You couldn't have been more wrong.
```

Display quoted passages in-game

```
      THE man ISA ACTOR AT street
      ...
        VERB ask
            WHEN act
            DOES ONLY
            """Whatever you ask me, I won't tell you,"" the man says arrogantly."
        END VERB.
      END THE man.
```

Yields

```
"Whatever you ask me, I won't tell you," the man says arrogantly.
```

Begin the response to a command with a small letter

```
THE man ISA PERSON AT street
  VERB talk_to
    DOES "$$blah blah..."
  END VERB.
END THE man.
```

Yields

```
>talk to man
blah blah...

(instead of:
>talk to man
Blah blah...)
```

The interpreter usually prints the first letter of a quote with a capital letter if it’s written with a small letter. `$$` overrides this.

Change the style (appearance) of the text

Use the STYLE command. The five alternatives to use are:

```
STYLE normal.
STYLE emphasized.
STYLE alert.
STYLE preformatted.
STYLE quote.
```

The actual outcome of the text is hugely interpreter-dependent. Experiment and see the various effects to determine which effect best suits your purposes.

SCORING

Change the player's score

```
THE diamond ISA OBJECT AT vault
...
  VERB take
    DOES SCORE 10.
  END VERB.
END THE.
```

`There is no negative score. If you want to reduce the player's score, make a numerical attribute that can be increased and decreased:`

```
THE hero ISA ACTOR
  HAS 'score' 0.
  ...
END THE hero.
```

```
VERB 'score'
  DOES ONLY "Your score is" SAY 'score' OF hero. "."
END VERB.

THE cliff ISA LOCATION
  ...
  VERB jump_off
    DOES DECREASE 'score' OF hero BY 5.
  END VERB.
END THE cliff.
```

NARRATION GIMMICKS

Modify the prompt (e.g. "What do you want to do next, Bob? >")

1)

```
START AT kitchen.
  PROMPT "What do you want to do next, Bob? >".
```

Yields:

```
Kitchen
What do you want to do next, Bob? >_
```

2)

```
VERB enter_www_address
  DOES
    PROMPT "http://".
END VERB.
```

Yields:

```
http://_
```

3)

```
START AT kitchen.
  SCHEDULE prompt_change AT hero AFTER 15.

EVENT prompt_change
  PROMPT "Let's try this kind of prompt from now on >".
END EVENT.
```

4)

```
PROMPT
  DEPENDING ON RANDOM 1 TO 2
  = 1 THEN "What next?>".
  = 2 THEN "What now?".
  END DEPEND.
```

Ignore what the player typed (carry on the story nevertheless)

- Use the extension "Restricted verbs" available on the ALAN website.

- Declare the location where you wish to ignore the player's command as a RESTRICTED_LOCATION.
- Edit the restricted_message in the extension so that it is an empty string: "" (or "$$").
- See the list of verbs in the extension code and add any you feel are missing (e.g. LOOK, RESTART...).
- Schedule an event to trigger at the point where you wish to ignore the player commands:

```
      THE void ISA RESTRICTED_LOCATION
        ENTERED SCHEDULE event1 AT void AFTER 1.
      END THE.

      EVENT event1
        "$$This message displays instead of the usual response to the player's
command."
        SCHEDULE event1 AT void AFTER 1.
      END EVENT.
```

To stop ignoring commands, give the restricted_location the attribute 'NOT restricted'.

Change the person of the narrative (e.g. "She sees nothing special about the book.")

This can't be done easily at present. You should change all the verb responses, checks and error messages manually. An extension might appear later to accomplish this feat automatically.

Change the tense of the narrative (e.g. "He found nothing interesting there.")

This can't be done easily at present. You should change all the verb responses, checks and error messages manually. An extension might appear later to accomplish this feat automatically.

OUT-OF-GAME ACTIONS

Undo a command

This action is built-in and doesn't need any programming. (It can neither be edited nor cancelled by the game programmer.) The command to be issued by the player is always UNDO; it won't allow any synonyms.

Repeat a command

Alan doesn't currently recognize the `AGAIN` command; the player can't have a command repeated in-game without writing it in full again. There is a workaround: use the arrow keys of the keyboard to scroll through the previously issued commands; this is also mentioned in the library default response for the command `AGAIN`.

Pause the game

There is no actual way to pause a game (e.g. "Press any key to continue.") or to clear the screen. To achieve this effect, workarounds must be used. Here is one:

```
EVENT garden_dreaming
  MAKE hero asleep.
    "You fall asleep under a bush and start dreaming....
    $p$p$p$p$p$p$p$p$p$p$p$p$p$p$p"
  LOCATE hero AT dreamland.
END EVENT.

START AT garden.

MESSAGES MORE:
  IF hero IS asleep
    THEN "(Please press SPACE to continue.)"
    ELSE "<More>"
  END IF.
```

Allow saving the game

This is handled automatically by the standard library when the player types "save".

(The command to use to force the game to be saved is `SAVE`:

```
VERB 'save'
  DOES SAVE.
END VERB.)
```

Make automatic saves

Here a save prompt appears automatically when the hero enters a cave:

```
THE cave ISA LOCATION
  ENTERED
    SAVE.
  DESCRIPTION "..."
END THE cave.
```

(Further coding would be needed to make sure e.g. that the save prompt appears only the first time the hero enters the cave etc.)

Note: Automatic saves are not commonly used in interactive fiction.

Allow restoring the game

This is handled automatically by the standard library when the player types "`restore`".

The command to use to make a restore prompt appear is `RESTORE`:

```
VERB 'restore'
   DOES RESTORE.
```

```
END VERB.)
```

Restart a game

```
SYNTAX 'restart' = 'restart'.

VERB 'restart'
   DOES
      RESTART.
END VERB.
```

When the player types:

```
>restart
```

The game will ask him (by default):

```
"Are you sure (RETURN confirms)?"
Pressing RETURN (=ENTER) will then restart the game.
```

COMMENTS

Add comments to the code

Every line of code starting with two dashes (`--`) is regarded as a comment. When these two dashes appear in the middle of the line, everything after the dashes on that line will be regarded as a comment:

1)

```
-- This is a comment.
```

2)

```
THE cave ISA LOCATION -- don't forget to add a description later!
   DESCRIPTION "..."
END THE cave.
```

Add comments in the game

If you want to make comments in-game, for example for a transcript file, use a semicolon (;) before the intended comment:

E.g.

```
        > wipe snow
        You wipe out the snow, which reveals a compass rose carved into the
ice.

        >;That was clever!
```

RUNTIME MESSAGES

Change the default runtime messages such as "You can't go that way." or "I don't know the word 'taek'."

```
MESSAGE NOWAY: "There is nothing of interest in that direction."

MESSAGE UNKNOWN_WORD: "The word '$1' is not needed in this story."
```

All default messages are listed in the manual and in the library file 'messages.i'.

Use parameters in runtime messages

When you look through the list of default error messages, you'll notice that some of them contain parameters: "I don't know the word '$1'.", "$+1 contains", etc. The parameters vary from message to message, put they can be referred to with "parameter1", "parameter2", etc. These parameters can then be used like in verb definitions.

```
MESSAGE NO_SUCH: "I can't see any $1 here."
  IF parameter1 = keys
    THEN "I think you left them at the office."
  ELSIF parameter1 = flashlight
    THEN "You forgot to take it with you from the bedroom."
         END IF.


MESSAGE HAVE_SCORED: "You have scored $1 points out of $2."
  IF parameter1 < 5
    THEN "That gives you the rank Beginner."
  ELSIF parameter1 BETWEEN 5 AND 10
    THEN ...
  ELSIF parameter2 / 2 > parameter1
    THEN "You haven't even reached halfway yet!"
      ELSE...
  END IF.


 MESSAGE UNKNOWN_WORD:   IF parameter1 = "d*mn" OR parameter1 = "sh*t" OR
parameter1 = "f***"
                  THEN "No dirty language please!"
             ELSE "I don't know the word '$1'."
               END IF.
```

Note: The `CHECK-ELSE-DOES` construction is not possible in runtime message definitions.

IMPORTING OTHER FILES

Import the standard library

Download the library from the ALAN website. The library contains four files.In your game code, declare:

```
IMPORT 'classes.i'.
IMPORT 'locations.i'.
IMPORT 'messages.i'.
IMPORT 'verbs.i'.
```

All these files will have to be in the same folder as your game code file. You can also just:

```
IMPORT 'library.i'.
```

which is a file in the library distribution package which in its turn imports all the four library files.

Import extensions

```
IMPORT 'ext1.i'.
```

This file would have to be in the same folder as your game file.

Import your own supporting code files

```
IMPORT 'gamefile2.i'.
```

- This file would have to be in the same folder as your game file.
- Don't give the supporting game file the extension .alan - reserve this extension to your main game file!
- It is fine for the supporting file to have an extension such as .txt. .i, etc.

USING MULTIMEDIA

Add images to your game

```
SHOW 'pic1.jpg'.
```

An image file must be in the same folder as the other game code files when you compile the game.

E.g.

```
THE boy ISA ACTOR AT street
 ...
VERB examine
  DOES ONLY SHOW 'boy1.jpg'.
END VERB.
END THE boy.
```

Add sounds to your game

```
PLAY 'sample1.aiff'.
```

A sound file must be in the same folder as the other game code files when you compile the game.

Ensure that the images and sounds are displayed in the distributed game

When you compile a game that uses multimedia, two files are created, 'mygame.a3c' and 'mygame.a3r'. Make sure that both files are included in the distributed game. (The original sound or image files don't have to be included.)

START SECTION

Determine the starting location of the game

1)

```
START AT forest_clearing.
```

2)

```
START AT kitchen.
  SCHEDULE burning_smell AT kitchen AFTER 3.
```

- START AT must always refer to a location instance.
- This must always be the **last** section in your game code - don't put it, for example, in the beginning of the code!
- The start section can contain some other sections, such as SCHEDULE that triggers events, or MESSAGE statements.
- It can also contain some introductory text to be displayed before the first location.

Add an introductory text before the first location

```
      START AT kitchen.
        "This is the introductory text that is shown before the first location,
the kitchen in this example."
```

Display no location name or description at the start of a game

```
      THE kitchen ISA LOCATION
        NAME ''
      END THE kitchen.

      START AT kitchen.
        "Only this (optional) text and no location at all will display at the
start of the game in this example."
```

The hero will still be at the kitchen at the start, even if the location is not mentioned or described.

Display the game title together with e.g. info about the author, version and copyright issues at the start of the game

E.g.

```
      START AT kitchen.
        "One day, when you planned to make a delicious dinner, you noticed
that something crucial was missing..."
        STYLE alert. "$pThe Mystery Of The Missing Salt Dispenser" STYLE
normal.
        "$nby An Author"
        "$nVersion 1"
        "$nProgrammed with ALAN version 3 beta 2
        $nAll rights reserved.$p"
```

Use INITIALIZE

You can use `INITIALIZE` within instances to make an instance behave in a certain way from the start of the game without using the `START` section:

```
THE dice ISA OBJECT AT livingroom
  HAS value 0.
  DESCRIPTION ...
  INITIALIZE
  SET value OF dice TO RANDOM 1 TO 6.
  VERB examine
    DOES ONLY "The dice is showing the value" SAY value OF dice. "."
  END VERB.
END THE dice.
```

You cannot declare e.g. "`HAS value RANDOM 1 TO 6.`" That's why using `INITIALIZE` is the way to go here.

When you examine the dice, it will have the random value set at the start of the game by `INITIALIZE`. You can also e.g. make actors act according to scripts using `INITIALIZE`:

```
 ADD TO EVERY ACTOR
  INITIALIZE
    USE SCRIPT walking_randomly_round_game_map FOR THIS.
END ADD TO.
```

(Here, every actor would act out the same script from the start of the game.)

ENDING THE GAME

End the game

```
QUIT.
```

This will by default yield

```
"Do you want to RESTART, RESTORE, QUIT or UNDO? >"
```

You can change this default by using e.g.

```
MESSAGE QUIT_ACTION: "RESTART/RESTORE/QUIT/UNDO?"
```

Code an ending (winning, losing) message to the game

1)

```
VERB jump
   DOES
     IF hero AT cliff_edge
       THEN "That wasn't such a clever idea."
         $p$p$p* * * You have died. * * *$p"
         QUIT.
       ELSE "You make a jump."
     END IF.
END VERB.
```

2)

```
EVENT winning
  "Opening the treasure chest, you see a wealth of jewelry and gold coins.
Congratulations! You have solved the game!"
  QUIT.
END EVENT.

THE treasure_chest ISA OBJECT AT cave
...
  VERB open
     DOES ONLY
       SCHEDULE winning AT cave AFTER 0.
  END VERB.
END THE treasure_chest.
```

3)

```
 MESSAGE QUIT_ACTION
   IF hero IS killed
     THEN "$p$p$p* * * You have died. * * *$p"
     ELSE "$p$p$p* * * You have won! * * * $p"
   END IF.
   "Would you like to RESTART, RESTORE, QUIT or UNDO? >"
```