The ALAN Adventure Language

Author's Guide

Preliminary!

Version 3.0beta2

This version of the authors guide was printed on August 14, 2011

Table of Contents

1 INTRODUCTION....4

1.1 A note on the Beta Version....5

1.2 Programmer's Pitch....5

1.3 To the Reader....5

2 WRITING WITH ALAN....8

2.1 Locations....8

A Simple Location....8

Interconnecting Locations....9

Doors between Locations....9

Locations with common scenery....11

2.2 Objects....11

Creating an Object....11

2.3 Actors....13

People versus Monsters....13

Making Actors Act....15

Conversing with Actors....15

Specialized Actors....15

2.4 Player Interaction....15

Restricting Player Reference To Some Objects....15

Easing Player Input....16

2.5 Common mistakes....16

1 INTRODUCTION

Text adventures or, using a more appropriate term, interactive fiction, is a form of computer game which has many things in common with fiction in book form, role-playing games and puzzle-solving. To create a high quality interactive fiction game, you need to be more of an author than a games programmer.

Alan is a special purpose computer language specifically designed to make it very easy to create such adventure games requiring only limited programming skills.

This guide is written for you that are attempting to write an Alan game and want some guidance on how to do that.

You will find snippets and examples in here. There is more complete examples and tutorials on the Alan web site, *http://www.alanif.se*, where you also can find downloads, more information about Alan and how to get in touch with the authors.

If you find interesting ways to implement game features, which are not covered here, or even better ways of doing what is described here, we'd love to hear from you!

Happy authoring!

2 WRITING WITH ALAN

This chapter will show you the basics of using the Alan features to write a small adventure. It is not a complete adventure. Instead, it will show you examples on how to implement commonly encountered needs.

The source examples are mostly excerpts from what is needed to get a complete game; this is indicated by the leading and trailing ellipses. Examples that are made by changing or adding from a previous example emphasize the differences.

More details about each construct are available in the Error: Reference source not found on page Error: Reference source not found.

2.1 Locations

A Simple Location

A very simple location would still have a description. The game should probably begin at this simple location.

```
The starting_place Isa location.
  Description
    "This place looks like the place where everything
     begins."
End The starting_place.
Start At starting_place.
```

This is a complete game. Try it.

Interconnecting Locations

Add another location and then we can connect them with exits.

```
…
The starting_place Isa location.
  Description
    "This place looks like the place where
      everything begins."
  Exit w To other_location.
End The starting_place.
…
```

Do the same with the other location; otherwise, we cannot travel back.

Doors between Locations

Sometimes we want to place an obstacle in the way of the player. One, perhaps to common, is a door that hinders the free passage. To implement this we need a door object.

```
…
The door Isa object At starting_place.
  Description
    "There is a door to the west."
End The door.
…
```

If you try this you will find that the door shows, but doesn't hinder much. We must 1) make the door open and close, and 2) check that it is open before allowing passage. Let's take item one first.

```
…
The door Isa object At starting_place.
  Is closed.
  Description
    "There is a door to the west."
    If door Is closed Then "It is closed."
    Else "It is open."
    End If.
End The door.
…
```

The new lines add an attribute that stores the open/closed state of the door. The player cannot see the attributes so we have to show him the state by adding a conditional messages of our choosing.

Then we need to prohibit traversal when the door is closed.

```
…
The starting_place Isa location.
  Description
    "This place looks like the place where everything
     begins."
  Exit w To other_location
    Check door Is Not closed
      Else "You can not walk through a closed door."
  End Exit.
End The starting_place.
…
```

Of course, we also need the player to be able to open it. This is requires a verb for the door.

```
…
The door Isa object At starting_place.
  Is closed.
  Description
    "There is a door to the west."
    If door Is closed Then "It is closed."
    Else "It is open."
    End If.
  Verb open
    Does
      Make door Not closed.
      "You open the door."
  End Verb.
End The door.
…
```

You can implement "close" in a similar way. If you run this, you will notice that there is no door in the other room. In fact, there isn't even a problem getting back even if the door would be closed. You can find tips on how to fix this in Error: Reference source not found on page Error: Reference source not found.

Locations with common scenery

There are a couple of ways to provide scenery. By scenery, we mean instances that are available but not important to the game. Often these are common to a region, or set of locations in the game, e.g. outdoors you will always find the ground and the sky. The easiest way to do this is to create regions of locations using the nested location feature.

This feature simply allows locations to be located at other locations, in effect enclose them by an "outer" location. The effect of this is simply that instances present in an outer location are reachable from the inner. Instances in an outer location are not automatically described.

This can of course be expanded further but a simple example would look like:

```
The outdoor Isa location
End The outdoor.

Every outdoorScenery Isa object At outdoor
End Every outdoorScenery.

The sky Isa outdoorScenery
End The sky.

The ground Isa outdoorScenery
End The ground.

Every outdoorLocation Isa location At outdoor
End Every outdoorLocation.
```

2.2 Objects

Creating an Object

In the current version of Alan all instances are static, in the sense that they are all created when the game starts. No new instances can be created or destroyed during the game. Nevertheless, there are ways to make it appear so to the player.

The stunt of creating a single object is easily achieved by just moving it from a storage location.

```
…
The storage Isa location
End The storage.

The coin Isa object At storage
End The coin.
…
  "As you rummage around among the leaves you see something
glimmering faintly."
  Locate coin Here.
…
```

Making objects disappear, or even convert into something else, is as easy. Move the disappearing object to your storage location, or exchange it for its morphed state.

```
  "As you try to pat the squirrel you realize that it actually is a
mongoose!"
  Locate squirrel At storage.
  Locate mongoose Here.
```

If you want the transformed object to keep some of the properties of the original, such as a name, you just need to make sure that they both have them. If they are complex you might consider creating a common super-class for those properties.

Sets of similar objects, such as a pile of coins, can be implemented by having the pile of coins as one object and the single coin(s) as separate instances. You just need to figure out how many separate instances are needed:

```
The pile Isa object At shop
  Name pile 'of' coins
End The pile.

The coin1 Isa coin At limbo
  Name small coin
End The coin1.

The coin2 Isa coin At limbo
  Name small coin
End The coin2.
…
  Verb pick_up
  …
    Depending On Count Isa coin, In hero
```

```
      =0: "You pick up a coin." Locate coin1 In hero.
      =1: "You pick up another coin." Locate coin2 In hero.
      Else "You already have two coins. Enough already!"
  End Depend.
```

Note however, that by giving the objects exactly the same name, the player can not distinguish between them, which might create problems. Maybe giving them names like "first small coin", "second small coin" etc. might work.

Fluids are much trickier, as they merge and split uncontrollably. To some extent they can be managed in the same manner as sets.

2.3 Actors

People versus Monsters

Since people usually have names, we might want to use that both in player references and in game output. An actor, Mr. Andersson, would need to have a name that the player can use:

```
  The mr_andersson Isa actor
    Name mr andersson …
```

This allows the player to refer to Mr Andersson as in:

```
> ask mr andersson about neo
```

However, consider the simplistic implementation of the verb (details removed for clarity):

```
  Verb ask_about
    Does
      Say The act.
      "does not want to tell you anything about"
      Say The subj.
      ".”
  End Verb.
```

This implementation will answer with

```
The mr andersson will not tell you anything about the neo.
```

Note the lower case initials of both mr Andersson and Neo. To make this right we will have to give them proper spelled names for the game to use. You can do this simply by writing

```
The mr_andersson Isa actor
   Name mr Andersson …

The neo Isa actor
   Name Neo …
```

This will instead give the output

```
> ask mr andersson about neo
The mr Andersson will not tell you anything about the Neo.
```

But people are seldom talked about in definite form like that. You don't say, "See, there goes the John." (At least not if you mean that the person John is walking over there…) Instead, you leave out definite and indefinite articles. The Alan run-time system, and probably some of your own messages, will need to refer to an unknown instance in definite or indefinite forms. E.g.

```
You cannot do that to a ball.
```

This works with things and actors that are referred to by their occupation or looks. However, to make people with names have natural definite and indefinite forms, you need to set the definite and indefinite articles to empty strings. You (or a standard library) could define a sub-class of actor to do this to all people, or persons with names:

```
Every namedPerson Isa actor
  Indefinite Article ""
  Definite Article ""
End Every person.

The mr_andersson Isa namedPerson
   Name mr Andersson
End The mr_andersson.
```

```
The neo Isa namedPerson
  Name 'Neo'
End The neo.
```

This would allow the following interaction:

```
Office
Mr Andersson is here.

> talk to mr andersson
You can not talk to mr Andersson.

> ask mr andersson about neo
Mr Andersson knows nothing about Neo.
```

Making Actors Act

<scripts, descriptions, …>

Conversing with Actors

<verbs like ASK, TELL, actor status, memory …>

Specialized Actors

<actors that have attributes (hungry) and act on their own>

2.4 Player Interaction

Restricting Player Reference To Some Objects

<default local instances, class restrictions, …>

Easing Player Input

<synonyms, multiple syntaxes>

2.5 Common mistakes

<show how to interpret compiler messages and other problems which are caused by missing terminating string quotes, quoting including spaces, reserved words, events running where the hero isn't>