ASL Language Reference Manual

ASL is a language that allows users to quickly create a set of scripted actions and lines in the format of a play's script, and run the script with a simple representation of actors. As actions are simplified into simple directional movement, users are able to write short action algorithms. Images may be used to replace the default representation of actors or props.

1. Lexical Conventions

1.1 Introduction
Tokens in ASL include actor names, integers, keywords, strings, and events. Whitespace is used only as a token separator. Actors refers to the broad set of all objects, and actor names are therefore similar to an identifier without a type. Events refers to a time starting from 00:00 that contains a set of actions to take place at that moment in the script’s run-time.

1.2 Comments
Comments are supported as all characters between /* and */

1.3 Actor names
Actor names can only alphabetical, and multiple parts to the name must be connected by ‘_’, underscore.

1.4 Keywords
SceneStart
SceneEnd
Actor
Left
Right
Up
Down
ULeft
URight
DFLeft
DRight
Text

1.5 Strings
Strings are composed of any character between “ and “.

1.6 Events
Events begin with a time, “[xx:xx]”, where x is any integer and the maximum run time is 99 hours and 60 minutes. They are then followed by a series of actions, designated by {} brackets. Every following action is considered part of the last event until a new event time is parsed.

1.7 Integers
Integers will be comprised of the digits 0-9. Floating point is not supported.

2. Statements
2.1 Introduction

*Events* and *Event actions* make up the core of ASL scripting. Every action involves an *actor* and an *action*. *Actions* are functions that are called on the involved *actor*. Functions involve the calling of the 8 directional and text keywords.

2.2 Events

*Events* are statements that executed in sequential order according to their times. *Event actions* within an *event* are then executed in sequential order.

\[
\text{Event} = [ \text{x x : x x} ] (\text{Event action})^* \\
\text{Event action} = \{ \text{actor , action, string}_{optional} , \text{int}_{optional} \}
\]

*Events* last as the *action* dictates. Movement *actions* take the same length of time to complete. Calls to *Text* will display the text until the next *Event* unless a time in seconds is specified after. Having an int without a string is invalid.

2.3 Scenes

*Scenes* create a scope by organizing the script, separating the *Actors* involved in them. *Actors* are of local scope to the scene they are declared in. *SceneStart* signifies a fade in, and *StartEnd* a fade out.

2.4 Actors

*Actors* must be declared prior to being used in a scene. They are local in scope to the scenes they are declared in. *Actors* are comprised of a name, initial position in x and y pixel-coordinates, and an image file to represent them.

\[
\text{Actor(“Bob”,0,60,”/Bob.jpg”)}
\]

3. Functions

3.1 Introduction

*Left*

*Right*

*Up*

*Down*

*ULeft*

*URight*

*DLeft*

*DRight*

*Text*

The above keywords compose functions. The 8 directional calls move the actor associated with the action call in that direction in sequential order. Using *Text* will display the associated text, if any, with the actor at that time.

3.2 Function Declarations

Function declarations are to be done first.

\[
\text{Function(“name”,action,...,action)}
\]

3.3 Function Invocations
Using a function in an event action:
Function(“hop”,Up,Right,Down)
[00:30] {Bob,hop}

4. Sample Program

Function(“Hop”,up,down)  
Function(“HopRight”,up,right,down)  
Function(“HopLeft”,up,left,down)  
Function(“DanceM”,up,right,left,left,right,up)  
Function(“DanceF”,up,right,left,left,right,down)  
SceneStart  
Actor(“Jack”,50,50,”/Jack.jpg”)  
Actor(“Box1”,50,60,”/OpenBox”)  
Actor(“Box2”,60,60,”/OpenBox”)  
[00:05] {“Jack”,”Hop”}  
[00:07] {“Jack”,”HopRight”} {“Jack”,”Say”,”Now I go into this box!”} 3  
[00:11] {“Jack”,”Say”,”The End”} 3 {“Jack”,”Down”}  
SceneEnd