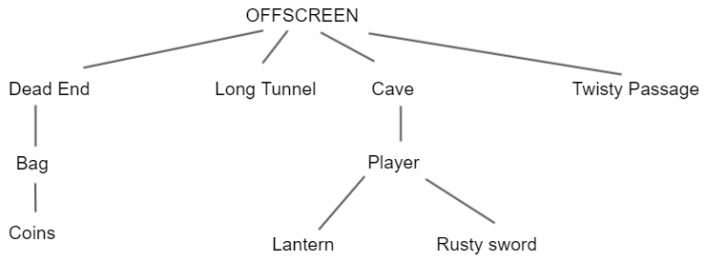
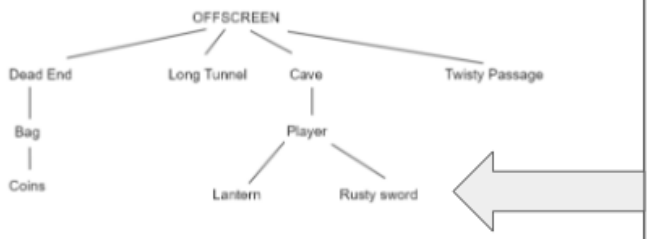
Lantern Quick Reference

World Model

The world is organized as a tree of objects. **Offscreen** is the root of the tree. Every object has a location attribute, which stores where it is in the tree.



Attributes and Properties



Attributes	Properties (1 or 0)
location description initial_description n,s,e,w,up,down,in,out ne,se,nw,sw mass	portable container supporter scenery wearable beingworn door open openable lockable locked lit user1,2,3,4

Every object has the following attributes, which can be accessed using the . operator,

Attribute	Purpose
location, parent, holder	Where this object is in the world
description	The description of the object when examined
initial_description	The text used when listing that object out in a room
n,s,e,w,ne,nw,se,sw, up,down,in,out	Which room the object is connected to in that direction

mass	How much it weighs. By giving objects mass, you can limit how much the player can carry. 10 is the default max the player can carry, but you can change it.

Properties

Every object has the following properties, which can be accessed using the . operator.

Property	**Purpose**
portable	Can the player 'take' this object?
scenery	Is this an invisible scenery object?
container	Can you put things in this object?
supporter	Can you put things on this object?
door	Is this object a door?
openable	Can this object be opened or closed?
open	Is this object open?
lockable	Can this be locked or unlocked?
locked	Is it actually locked?
wearable	Is this something the player can wear?
worn	Is it being worn?
lit	Is his object emitting light (used to determine visibility)
user1, user2, user3, user4	Properties for you to use as needed

Built in functions

describe(object)	Prints an object's description
print(“”);	Prints a new line
print(“Hi”);	Prints *Hi*
println(“Hi”);	Prints *Hi* followed by a new line
printvar(*health*);	Prints the value of *health*
printname(obj);	Prints the name of an object
light_check();	Sets canSee to 1 if player can see, otherwise 0
look();	Prints the player's room description

look_in(object)	Prints out the contents of the object
ask();	Asks user to type a line. The result is stored in the variable *answer.*
anykey();	Waits for use to press a key
cls();	Clears the screen
rand(15);	*Returns a random number between 0 and 15. (Works up to 255)*
stop();	Used to make a verb check fail and prevent sentences from running

Variables

Variables can store values from **0** to **255**. You can create your own on the variables tab. Variables in Lantern are **global**, meaning they can be accessed from any function.

Built-in variables

Variable	Purpose
health	Stores the player's health. (Starts at 100)
score	Used for the score. (Starts at 0)
moves	How many turns the game has been going on. (Starts at 0)
turnsWithoutLight	Can be used to kill the player if they're lost in the dark too long
answer	Stores the answer typed in as a result of ask();
gameOver	Whether the game is over or not. (Starts as 0)
noun1	The first noun in the last command (255 if not entered)
noun2	The second noun in the last sentence (255 if not entered)
maxWeight	How much the player can carry (Defaults to 10)
invWeight	Weight of the player's inventory
canSee	Set by light_check()

Checks, Events, Functions, and Sentences

Checks are used to determine if a command should be carried out.
Functions contain code that will run when the user types a command.
Sentences link a command the player types to a *function* to execute.
Events are functions which run every turn a command was successfully processed.

Sentence types

Before: Doing something before the "Instead" or built in response is run
Example: before actually moving east, print "You trudge east up the hill..."

Instead (most common): provide or override a response to a command

After: Used for printing the result of the command or taking additional action.
Example: after taking the gold coin, print "You stealthily pocket the gold coin."

Lantern Coding Syntax (Based on C)

Tutorials: http://textadventure.net contains many, many tutorials...use them!

Operators and symbols

Symbol	Meaning	Example
//	comment operator	//this just a note in the code
;	End of statement	(see below for examples)
=	Assignment (set left side equal to right)	health = 100; //set health to 100
.	Access a property/attribute of an object	torch.lit = 1; //now torch is lit
++	Add 1 to a variable	health++; //add 1 to health
--	Subtract 1 from a variable	health--; //subtract 1 from health
+=	Add the right side to the left side	score += 10; //add 10 to score
-=	Subtract the right side from the left side	health -= 20; //subtract 20 from health

Comparison Operators

a == b compare and test for equality a != b compare and test for inequality a < b test if a is less than b	a > b test if a is greater than b a >= b test if a is greater than or equal to b a <= b test if a is less than b

Logical Operators

&& means ‘and’	if (score == 100 && health == 100)
	{
	  println(“Your score **and** health are 100”);
	}

|| means ‘or’	if (timeLeft == 0 || health == 0)
	{
	  println(“You are out of time **or** health”);
	}

Control Flow

If statement (if + tab + tab)	**If / else statement (ie + tab + tab)**

if (*expression*)
{
  //statements
}
	if (*expression*)
	{
	  //statements
	}
	else if (*some other expression*)
	{
	  //statements
	}
	else
	{
	  //statements
	}

Keyboard shortcuts

Letters followed by tab key twice	Result
if	Creates an if statement
else	Creates to an else statement
ei	Creates an else if statement
pl	Expands to *player*
of	Expands to *offscreen*
.d	Expands to *.description*
.e	Expands to .lit (emitting light)
.h	Expands to *.holder*
.i	Expands to *.initial_description*
.l	Expands to *.location*
.p	Expands to *.parent*
.o	Expands to .o*pen*
.w	Expands to .worn
pr	Expands to *println(“”);*
pv	Expands to printvar( );
pn	Expands to printname( );