Instead Engine CheatSheet

The Head

Any sequence of characters starting with a double-dash on the same line is a Lua comment. INSTEAD tags are comment lines of the following form:

```
-- $TagName: A string of UTF-8 characters$

-- $Name: the Most interesting game!$
-- $Version: 0.5$
-- $Author: Anonymous fan of text adventures$
-- $Info: This is a remake of a classic\nZX Spectrum
game.$
```

The Object

```
obj {
  nam = 'table';
  dsc = 'In the {table}.';
  act = 'Hm... Just a table...';
  tak = 'I took the corner of the table';
  inv = 'I hold the corner of the table.';
  };
```

The Room

```
room {
  nam = 'main';
  title = 'Start of adventure';
  disp = "Main room";
  dsc = [[You are in a large room.]];
  way = { 'hall', 'street' };
  obj = { 'table', 'armchair' };
}
```

Dialogue

```
dlg {
  nam = 'conversation';
  -- in the dialogues typically do not require inventory
  noinv = true;
  title = [[the Conversation with the seller]];
  enter = [[I asked the seller.]]];
  phr = {
    { 'You have beans?', '-- No.'},
```

The connection modules

require "module name" or loadmod "module name"

```
require "click" -- plug-in click;
require "fmt" -- some formatting functions
fmt.para = true -- enable the paragraphs (indents)
```

The Modules

- 'dbg' module debugging.
- 'click' – module intercept mouse clicks on the image of the scene;
- 'prefs' module settings (data warehouse settings);
- 'snapshots' – a plugin to support snapshots (for kickbacks of game situations);
- 'fmt' module of withdrawal;
- 'theme' – the office theme on the fly;
- 'noinv' module to operate the equipment;
- 'key" - module event processing of the operation keys;
- 'timer' - timer;
- 'sprite' – module for working with sprites;
- 'snd' module audio;
- 'nolife' module locking methods life;

The List Attributes

List attributes (such as 'way' or 'obj') allow you to work with its content with a set of methods.

```
room {
  nam = 'fridge';
  frost = std.list { 'ice cream' };
}
ways():disable() -- disable all transitions
```

- disable() - disables all objects in the list;
- enable() - enables all objects of the list.
- close() - close all objects of the list.
- open() - open the list objects;
- add(object|name [position]) - add an object;
- for_each(function, args) - to call for each object feature arguments;
- lookup(name/tag or object) is object search in the list. Returns the object and the index;

Default handlers

```
game.act = 'Not running.';
game.use = 'It does not help.';
game.inv = 'Why?';
```

Global variables

```
global { -- definition of global variables
  global_var = 1; -- number
  some_number = 1.2; -- number
  some_string = 'string';
  know_truth = false; -- a Boolean value
  array = {1, 2, 3, 4}; -- array
}
```

Constants

```
const {
  A = 1;
  B = 2;
}
const 'Aflag' (false)
```

Declaration

```
declare {
  A = 1;
  B = 2;
}
declare 'Z' (false)

declare 'test' (function()
  p "Hello world!"
end)

global 'f' (test)
```

Helper functions

```
function mprint(n, ...)
  local a = {...}; -- temporary array with the arguments
to the function
  p(a[n]) -- get the n-th element of the array
```

```
    { 'You have chocolate?', '-- No.'},
    { 'You have a brew?', '-- Yes',
      { 'How much is it worth?', '-- 50 rubles.' },
      { 'He is cold?', '-- Fridge was broken.',
        { 'Take two!', 'Left one.',
          { 'Give me one!', function() p [[OK!]]; take
'brew'; end };
        }
      }
    }
  }
}
```

Standard library functions

[wh] - indicates an optional parameter.

- include(file) - file to include in the game; include "lib" -- will include the lib file.lua from the current directory with the game;
- loadmod(module) to connect the module of the game; loadmod "module" -- will include the module module.lua from the current directory;
- rnd(m) - a random integer value from '1' to 'm';
- rnd(a, b) - a random integer value from 'a' to 'b' where 'a' and 'b' are integers >= 0;
- rnd_seed () - set seed of random number generator;
- p (...) output a string to the buffer handler/attribute (with a space at the end);
- pr (...) output a string to the buffer handler/attribute "as is";
  pn (...) output a string to the buffer handler/attribute (with a newline at the end);
- pf(fmt, ...) - output formatted string to the buffer handler/attribute;
  local text = 'hello';
  pf("String: %q: %d\n", text, 10);
- pfn(...)(...)... "line" - creating a simple handler; This feature simplifies the creation of simple handlers:
  act = pfn(walk, 'bathroom') "I decided to go to the bathroom.";
  act = pfn(enable, '#transition') "I noticed a hole in the wall!";
- obj {} - create object;
- stat {} - create status;
- room {} - create a room;
- menu {} - create a menu;
- srch(name/tag or the object) - the search for a visible object in the list;
- empty() - returns true if the list is empty;
- zap() - clear the list;
- replace(what, what) - replace the object in the list;
- cat(list, [position]) - add the contents of the list into the current list position;
- del(name/object) - delete object from the list.

CThere are functions that return the objects lists:

- inv([player]) - return the player's inventory;
- objs([where]) - return objects of the room;
- ways([room]) - return transitions of the room.

Of course, you can refer to the lists directly:

```
pl.obj:add 'нож'
```

The objects in the lists are stored in the order in which they add. However, if the object is present numeric attribute pri he plays the role of priority in the list. If pri is not specified, the value priority 0 is considered. Thus, if you want some the object was first on the list, give priority pri < 0. If the end of the list – > 0.

```
obj {
  pri = -100;
  nam = 'thing';
  disp = 'Very important item';
  inv = [[Careful with this subject.]];
}
```

Functions that return objects

- the characters [ ] describe the optional parameters;
- 'what' or 'where' - means an object (including the room) is specified by the tag name or a variable reference;

- '_(what) ' - get the object;
- 'me()' returns the current object to the player;
- 'here()' returns the current scene.
- 'where ()' returns a room or an object which is the specified object if the object is in multiple places, you can pass in a second parameter -- a Lua table that will be added these objects;
- 'inroom ()' similar to where(), but returns the room in which the facility is located (this is important for objects in the objects);
- 'from([where])' returns the last room the player goes into given room. Optional parameter -- to the last room not for the current room, and for a given;

```
end
```

The "Player" Object

By default, the attribute 'obj', the player represents the inventory.

```
game.player = player {
  nam = "Basil";
  room = 'kitchen'; -- the starting room of the player
  power = 100;
  obj = { 'Apple' }; -- let's give him an Apple
};
```

The "World" Object

The name of this object the 'game'. There is a reference variable, also called game.

```
game.act = 'does Not work.';
game.inv = 'hmm ... Odd thing...';
game.use = 'does Not work...';
game.tak = 'I don't need this...';
```

Object methods (obj)

- :with({...}) - set list obj;
- :new (...) constructor;
- :actions(type, [value]) - set/read the number of event object the preset type;
- :inroom([{}]) - in which room (room) the object is located;
- :where([{}]) - what the object (objects) is a object;
- :purge() - removes the object from all lists.
- :remove() - remove object from all objects/rooms /equipment;
- :close ()/open() - close/open;
- :disable()/:enable() - disable/enable;
- :closed() -- returns true if closed;
- :disabled() -- returns true if disabled;
- :empty() -- returns true if empty;- :save(fp, n) -- save function;
- :display() -- display function in the scene;
- :visible() -- returns true if it is visible;
- :srch(w) - the search for the visible object;
- :lookup(w) - search any object;
- :for_each(fn, ...) -- iterator over the objects;
- :lifeon()/:lifeoff() -- add/remove from the list of living;

- dlg {} - create a dialogue;
- me() - returns the current player;
- here() - returns the current scene.
- from([w]) - returns the room from which the transition to your current scene.
- new(constructor, arguments) - creates a new dinamicheskogo object (to be described later);
- delete(w) - deletes the dynamic object;
- gamefile(file, [reset?]) - load dynamically the file with the game;
  gamefile("part2.lua", true) -- reset the game state (remove objects and variables), load part2.lua and start with the main room.
- player {} - create a player;- dprint(...) - debug output;
- visits([w]) - the number of visits to the bathroom (or 0 if visits);
- visited([w]) - the number of visits to the room, or false if not visits was;
  if not visited() then
    p [[it's my first time.]]
  end
- walk(w, [Boolean exit], [enter Boolean], [Boolean to change from]) - transition in the scene;
  -- unconditional jump (to ignore onexit/the onenter/exit /enter);
  walk('end', false, false)
- walkin(w) is a transition in the scene (without calling exit/onexit current);
- walkout([w], [dofrom]) - return from sub-scene (without calling enter/the onenter);
- walkback([w]) - a synonym walkout([w], false);
- _(w) - receiving object;
- for_all(fn, ...) - to perform the function for all arguments;
  for_all(enable, 'window', 'door');
- seen(w, [where]) - search for the visible object;
- lookup(w, [where]) is a search object;
- ways([where]) - get list of transitions;
- objs([where]) - get the list of objects;
- search(w) - search for the player object;
- have(w) - search for items in the inventory;
- inroom(w) - the return of the room/rooms, in which the object resides;
- where(w, [table]) - return the object/objects in which the object resides;
  local list = {}
- 'seen(what [, where])' returns an object or transition, if itis present and can see, there is a second optional parameter -- select the scene or object/list in which to search;
- 'lookup(what, [where])' returns an object or transition, if it there is in the scene or object/list;
- 'inspect ()' returns the object if it is visible/available on stage. The search is performed for transitions and objects, including, in the object of the player;
- 'have ()' returns the object if it is in the inventory and not disabled;
- 'live ()' returns the object if it is present among the living objects (described below);

List of attributes and handlers for bjects and rooms (obj, room)

- nam - attribute;
- tag - attribute;
- ini - handler, called for an object / room during construction game world, can only be a function;
- dsc - attribute, called to output the description;
- disp - attribute, information about an item in the inventory or a room in way list;
- title - attribute of the room, the name of the room displayed when found inside this room;
- decor - attribute of the room, called to display a description of the static decorations in scene;
- nolife - room attribute, do not call live object handlers;
- noinv - room attribute, do not show inventory;
- obj - attribute, list of nested objects;
- way - attribute of room, list of rooms to walk;
- life - handler, called for "live" (background) objects;
- act - object handler, called when acting on a scene object;
- tak - handler for taking an object from the scene (if act is not specified);
- inv - object handler, called when acting on an inventory object;
- use(s, on what) - object handler, called when using inventory item on another item;
- used (s, that) - object handler, called before use when using this item (passive form);
- onenter(s, from where) - room handler, called when
- :live() -- returns true if the list alive.

Room methods (room)

In addition to the methods of obj, added the following methods:

- :from() -- where I came into the room;
- :visited() -- was there a room you visited earlier?;
- :visits() -- number of visits (0-if not);
- :scene() -- display scene (no objects);
- :display() -- display of objects in the scene;

Dialogs metods (dlg)

In addition to the methods room, added the following methods:

- :push(phrase) - go to phrase, remembering it in the stack;
- :reset(phrase) - same, but reset the stack;
- :pop(phrase) -- return stack;
- :select(phrase) -- select current phrase;
- :ph_display() -- display the selected phrase;

The game world (game object)

In addition to the methods of obj, added the following methods:

- :time([v]) -- set/get the number of game cycles;
- :lifeon([v])/:lifeoff([v]) -- add/remove an object from the list alive, or enable/disable live listings globally (if not specified argument);
- :live([v]) -- check the activity of the life object;
- :set_pl(pl) -- switch player
- :life () - the iteration of live objects;
- :step() -- tact of the game;
- :lastdisp([v]) - set/get the latest output;
- :display(state) -- display output;
- :lastreact([v]) - set/get the latest reaction;
- :reaction([v]) -- set/get current response;
- :events(pre, bg) -- set/get events of live objects;
- :cmd(cmd) -- the command INSTEAD;

Player (player)

In addition to the methods of obj, added the following methods:

- :moved () - the player made a move in the current cycle of the game;
- :need_scene([v]) - need to render the scene in this cycle;

local w = where('Apple', list)
-- if the Apple is in more than one place, then list will contain an
-- array of these places. If you only need one location, then:
where 'Apple' -- will be enough

- closed(w) - true if the object is closed;
- disabled(w) - true if the object is off;
- enable(w) is to include an object;
- disable(w) - off object;
- open(w) - open;
- close(w) - close the object;
- actions(w, string, [value]) - returns (or sets) the number of actions of type t to an object w.
  if actions(w, 'tak') > 0 then -- object w was taken at least 1 time;
  if actions(w) == 1 then -- act it w was called 1 times;
- pop(tag) - return to the last branch of the dialog;
- push(tag) - the transition to the next branch of dialogue
- empty([w]) - empty right branch of the dialogue? (or object)
- lifeon(w) - add an object to the list of the living;
- lifeoff(w) - to remove an object from the list of the living;
- live(w) - object is alive?;
- change_pl(w) - change a player;
- player_moved([pl]) is the current player moved in this manner?;
- inv([pl]) - get a list of the inventory;
- remove(w, [wh]) - delete the object from object or room; Removes object obj from list and way (leaving all the rest, for example, game.lifes);
- purge(w) - destroy the object (from all lists); Removes the object from all lists in which it is present;
- replace(w, ww, [wh]) is to replace one object to another;
- place(w, [wh]) is to put the object in the object/room (removing it from the old object/rooms);
- put(w, [wh]) - put object without removing it from the old location;
- take(w) - pick up object;
- drop(w, [wh]) - to throw object;
- path {} - create a transition;
- time () is the number of moves from the beginning of the game.

entering the the room, can cancel when returning false;

- enter(s, where) - room handler, is called after a successful entering the room;
- onexit(s, where) - room handler, called when leaving room, can cancel when returning false;
- exit(s, where) - room handler, is called after a successful exit from the room.

List of attributes and handlers for game world (game)

- use (s, what, on what) - handler, default action for the use of the object;
- act(s, that) - handler, default action while object click;
- inv(s, that) - handler, default action on inventory click;
- on{use, act, tak, inv, walk} - handler, response before call appropriate handlers, can cancel the chain;
- after{use, act, tak, inv, walk} - handler, reaction after plaayer action.

- :inspect(w) - find an object (visible) in the current scene or really;
- :have(w) - search in the inventory;
- :useit(w) - use item;
- :useon(w, ww) -- use the item on the subject;
- :call(m, ...) -- calling a method of the player;
- :action(w) - action on the subject (act);
- :inventory() -- return the inventory (list, default is obj);
- :take(w) -- take the object;
- :walk/walkin/walkout -- transitions;
- :go(w) - team go (checking the availability of transitions);