Instead Engine CheatSheet

Заголовок

```
-- $Name: Самая интересная игра!$
-- $Version: 0.5$
-- $Author: Анонимный любитель текстовых приключений$
-- $Info: Это ремейк старой игры\nC ZX specturm$
```

Объект

```
obj {
  nam = 'стол';
  dsc = 'В комнате стоит {стол}.';
  act = 'Гм... Просто стол...';
  };
```

Комната

```
room {
  nam = 'main';
  title = 'Начало приключения';
  disp = "Главная комната";
  dsc = [[Вы в большой комнате.]];
  way = { 'hall', 'street' };
  obj = { 'ящик', apple };
}
```

Диалог

```
dlg {
  nam = 'разговор';
  -- в диалогах обычно не нужен инвентарь
  noinv = true;
  title = [[Разговор с продавцом]];
  enter = [[Я обратился к продавцу.]];
  phr = {
    { 'У вас есть бобы?', '-- Нет.'},
    { 'У вас есть шоколад?', '-- Нет.'},
    { 'У вас есть квас?', '-- Да',
```

Подключение модулей

```
require "click" -- перехват кликов мыши по картинке сцены;
require "fmt" -- некоторые функции форматирования
fmt.para = true -- включить режим параграфов (отступы)
```

Модули

- 'dbg' — модуль отладки;
- 'click' — модуль перехвата кликов мыши по картинке сцены;
- 'prefs' — модуль настроек (хранилище данных настроек);
- 'snapshots' — модуль поддержки снапшотов (для откатов игровых ситуаций);
- 'fmt' — модуль оформления вывода;
- 'theme' — управление темой на лету;
- 'noinv' - модуль работы с инвентарем;
- 'key'' - модуль обработки событий срабатывания клавиш;
- 'timer' - таймер;
- 'sprite' — модуль для работы со спрайтами;
- 'snd' — модуль работы со звуком;
- 'nolife' — модуль блокировки методов life;

Атрибуты-списки

Атрибуты-списки (такие как 'way' или 'obj') позволяют работать со своим содержимым с помощью набора методов

```
room {
  nam = 'холодильник';
  frost = std.list { 'мороженное' };
}
```

- disable() - отключает все объекты списка;
- enable() - включает все объекты списка;
- close() - закрыть все объекты списка;

Обработчики по-умолчанию

```
game.act = 'Не работает.';
game.use = 'Это не поможет.';
game.inv = 'Зачем мне это?';
```

Глобальные переменные

```
global { -- определение глобальных переменных
  global_var = 1; -- число
  some_number = 1.2; -- число
  some_string = 'строка';
  know_truth = false; -- булево значение
  array = {1, 2, 3, 4}; -- массив
}
```

Константы

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

Декларации

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

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

global 'f' (test)
```

Вспомогательные функции

```
      { 'А сколько он стоит?', '-- 50 рублей.' },
      { 'А он холодный?', '-- Холодильник
сломался.',
        { 'Беру два!', 'Остался один.',
          { 'Дайте один!', function() p [[Ок!]];
take 'квас'; end };
        }
      }
    }
  }
}
```

Функции стандартной библиотеки

- include(файл) - включить файл в игру;
- loadmod(модуль) - подключить модуль игры;
- rnd(m) - случайное целочисленное значение от '1' до 'm';
- rnd(a, b) - случайное целочисленное значение от 'a' до 'b', где 'a' и 'b' целые >= 0;
- rnd_seed(что) - задать зерно генератора случайных чисел;
- p(...) - вывод строки в буфер обработчика/атрибута (с пробелом в конце);
- pr(...) - вывод строки в буфер обработчика/атрибута "как есть";
- pn(...) - вывод строки в буфер обработчика/атрибута (с переводом строки в конце);
- pf(fmt, ...) - вывод форматной строки в буфер обработчика/атрибута;

- pfn(...)(...)... "строка" - формирование простого обработчика;
- obj {} - создание объекта;
- stat {} - создание статуса;
- room {} - создание комнаты;
- menu {} - создание меню;
- dlg {} - создание диалога;
- me() - возвращает текущего игрока;
- here() - возвращает текущую сцену;
- from([w]) - возвращает комнату из которой осуществлен переход в текущую сцену;
- new(конструктор, аргументы) - создание нового
- open() - открыть все объекты списка;
- add(объект|имя, [позиция]) - добавить объект;
- for_each(функция, аргументы) - вызвать для каждого объекта функцию с аргументами;
- lookup(имя/тег или объект) - поиск объекта в списке. Возвращает объект и индекс;
- srch(имя/тег или объект) - поиск видимого объекта в списке;
- empty() - вернет true, если список пуст;
- zap() - очистить список;
- replace(что, на что) - заменить объект в списке;
- cat(список, [позиция]) - добавить содержимое списка в текущий список по позиции;
- del(имя/объект) - удалить объект из списка.

Существуют функции, возвращающие объекты-списки:

- inv([игрок]) - вернуть инвентарь игрока;
- objs([комната]) - вернуть объекты комнаты;
- ways([комната]) - вернуть переходы комнаты.

```
pl.obj:add 'нож'
```

Числовой атрибут pri, играет роль *приоритета* в списке. Если pri не задан, значением приоритета считается 0. Таким образом, если вы хотите, чтобы какой-то объект был первым в списке, давайте ему приоритет pri < 0. Если в конце списка – > 0.

```
obj {
  pri = -100;
  nam = 'штука';
  disp = 'Очень важный предмет инвентаря';
  inv = [[Осторожней с этим предметом.]];
}
```

Функции, возвращающие объекты

- в символах [ ] описаны необязательные параметры;
- 'что' или 'где' - означает объект (в том числе комнату), заданный тегом, именем или переменной-ссылкой;

- '_(что)' - получить объект;
- 'me()' возвращает текущего объекта-игрока;
- 'here()' возвращает текущую сцену;
- 'where(что)' возвращает комнату или объект в котором

```
function mprint(n, ...)
  local a = {...}; -- временный массив с аргументами
к функции
  p(a[n]) -- выведем n-й элемент массива
end
```

Объект "Игрок"

По умолчанию, атрибут 'obj' у игрока представляет собой инвентарь.

```
game.player = player {
  nam = "Василий";
  room = 'кухня'; -- стартовая комната игрока
  power = 100;
  obj = { 'яблоко' }; -- заодно добавим яблоко в
инвентарь
};
```

Объект "Мир"

Имя такого объекта 'game'. Существует ссылка-переменная, которая также называется game.

```
game.act = 'Не получается.';
game.inv = 'Гм.. Странная штука..';
game.use = 'Не сработает...';
game.tak = 'Не нужно мне это...';
```

Методы объекта (obj)

- :with({...}) - задание списка obj;
- :new(...) - конструктор;
- :actions(тип, [значение]) - задать/прочитать число событий объекта заданного типа;
- :inroom([{}]) - в какой комнате (комнатах) находится объект;
- :where([{}]) - в каком объекте (объектах) находится объект;
- :purge() - удалить объект из всех списков;
- :remove() - удалить объект из всех объектов/комнат /инвентаря;
- :close()/:open() - закрыть/открыть;
- :disable()/:enable() - выключить/включить;

динамического объекта
- delete(w) - удаление динамического объекта;
- gamefile(файл, [сбросить состояние?]) - подгрузить динамически файл с игрой;
- player {} - создать игрока;
- dprint(...) - отладочный вывод;
- visits([w]) - число визитов в данную комнату (или 0, если визитов не было);
- visited([w]) - число визитов в комнату или false, если визитов не было;
- walk(w, [булевое exit], [булевое enter], [булевое менять from]) - переход в сцену;
- walkin(w) - переход в под-сцену (без вызова exit/onexit текущей комнаты);
- walkout([w], [dofrom]) - возврат из под-сцены (без вызова enter/onenter);
- walkback([w]) - синоним walkout([w], false);
- _(w) - получение объекта;
- for_all(fn, ...) - выполнить функцию для всех аргументов;
- seen(w, [где]) - поиск видимого объекта;
- lookup(w, [где]) - поиск объекта;
- ways([где]) - получить список переходов;
- objs([где]) - получить список объектов;
- search(w) - поиск доступного игроку объекта;
- have(w) - поиск предмета в инвентаре;
- inroom(w) - возврат комнаты/комнат, в которой находится объект;
- where(w, [таблица]) - возврат объекта/объектов, в котором находится объект;
- closed(w) - true если объект закрыт;
- disabled(w) - true если объект выключен;
- enable(w) - включить объект;
- disable(w) - выключить объект;
- open(w) - открыть объект;
- close(w) - закрыть объект;
- actions(w, строка, [значение]) - возвращает (или устанавливает) число действий типа t для объекта w.
- pop(тег) - возврат в прошлую ветвь диалога;
- push(тег) - переход в следующую ветвь диалога
- empty([w]) - пуста ли ветвь диалога? (или объект)
- lifeon(w) - добавить объект в список живых;
- находится заданный объект, если объект находится в нескольких местах, то можно передать второй параметр -- таблицу Lua, в которую будут добавлены эти объекты;
- 'inroom(что)' аналогично where(), но вернет комнату, в которой расположен объект (это важно для объектов в объектах);
- 'from([где])' возвращает прошлую комнату, из которой игрок перешел в заданную комнату. Необязательный параметр -- получить прошлую комнату не для текущей комнаты, а для заданной;
- 'seen(что, [где])' возвращает объект или переход, если он присутствует и видим, есть второй необязательный параметр -- выбрать сцену или объект/список в котором искать;
- 'lookup(что, [где])' возвращает объект или переход, если он существует в сцене или объекте/списке;
- 'inspect(что)' возвращает объект, если он виден/доступен на сцене. Поиск производится по переходам и объектам, в том числе, в объектах игрока;
- 'have(что)' возвращает объект, если он есть в инвентаре и не отключен;
- 'live(что)' возвращает объект, если он присутствует среди живых объектов (описано далее);

Атрибуты и обработчики объектов и комнат (obj и room)

- ini - обработчик, вызывается для объекта/комнаты при конструировании игрового мира, может быть только функцией;
- dsc - атрибут, вызывается для вывода описания;
- disp - атрибут, информация об объекте в инвентаре или комнаты в списке переходов;
- title - атрибут комнаты, название комнаты выводимое при нахождении внутри этой комнаты;
- decor - атрибут комнаты, вызывается для вывода описания декораций в сцене;
- nolife - атрибут комнаты, не вызывать обработчики живых объектов;
- noinv - атрибут комнаты, не показывать инвентарь;
- :closed() -- вернет true, если закрыт;
- :disabled() -- вернет true, если выключен;
- :empty() -- вернет true, если пуст;
- :save(fp, n) -- функция сохранения;
- :display() -- функция отображения в сцене;
- :visible() -- вернет true если считается видимым;
- :srch(w) -- поиск видимого объекта;
- :lookup(w) -- поиск любого объекта;
- :for_each(fn, ...) -- итератор по объектам;
- :lifeon()/:lifeoff() -- добавить/удалить из списка живых;
- :live() -- вернет true, если в списке живых.

Методы комнаты (room)

Кроме методов obj, добавлены следующие методы:

- :from() -- откуда пришли в комнату;
- :visited() -- была ли комната посещена ранее?;
- :visits() -- число визитов (0 -- если не было);
- :scene() -- отображение сцены (не объектов);
- :display() -- отображение объектов сцены;

Методы диалога (dlg)

Кроме методов room, добавлены следующие методы:

- :push(фраза) - перейти к фразе с запоминанием ее в стеке;
- :reset(фраза) -- то же самое, но со сбросом стека;
- :pop([фраза]) -- возврат по стеку;
- :select([фраза]) -- выбор текущей фразы;
- :ph_display() -- отображение выбранной фразы;

Методы игрового мира (game)

Кроме методов obj, добавлены следующие методы:

- :time([v]) -- установить/взять число игровых тактов;
- :lifeon([v])/:lifeoff([v]) -- добавить/удалить объект из списка живых, или включить/выключить живой список глобально (если не задан аргумент);
- :live([v]) -- проверить активность живого объекта;
- :set_pl(pl) -- переключить игрока;
- :life() -- итерация живых объектов;

- lifeoff(w) - убрать объект из списка живых;
- live(w) - объект жив?;
- change_pl(w) - смена игрока;
- player_moved([pl]) - текущий игрок перемещался в этом такте?;
- inv([pl]) - получить список-инвентарь;
- remove(w, [wh]) - удалить объект из объекта или комнаты; Удаляет объект из списков obj и way (оставляя во всех остальных, например, game.lifes);
- purge(w) - уничтожить объект (из всех списков); Удаляет объект из *всех* списков, в которых он присутствует;
- replace(w, ww, [wh]) - заменить один объект на другой;
- place(w, [wh]) - поместить объект в объект/комнату (удалив его из старого объекта/комнаты);
- put(w, [wh]) - поместить объект без удаления из старого местоположения;
- take(w) - забрать объект;
- drop(w, [wh]) - выбросить объект;
- path {} - создать переход;
- time() - число ходов от начала игры.

- obj - атрибут, список вложенных объектов;
- way - атрибут комнаты, список с переходами в другие комнаты;
- life - обработчик, вызывается для "живых" (фоновых) объектов;
- act - обработчик объекта, вызывается при действии на предмет сцены;
- tak - обработчик взятия предмета со сцены (если не задан act);
- inv - обработчик объекта, вызывается при действии на предмет инвентаря;
- use(s, на что) - обработчик объекта, вызывается при использовании предмета инвентаря на предмет сцены или инвентаря;
- used(s, что) - обработчик объекта, вызывается перед use при использовании предмета (страдательная форма);
- onenter(s, откуда) - обработчик комнаты, вызывается при заходе в комнату игрока, может запретить переход;
- enter(s, откуда) - обработчик комнаты, вызывается после успешного входа в комнату;
- onexit(s, куда) - обработчик комнаты, вызывается при выходе из комнаты, может запретить переход;
- exit(s, куда) - обработчик комнаты, вызывается после успешного выхода из комнаты.

Атрибуты и обработчики >игрового мира (game)

- use(s, что, на что) - обработчик, действие по умолчанию для использования предмета;
- act(s, что) - обработчик, действие по умолчанию при применении предмета сцены;
- inv(s, что) - обработчик, действие по умолчанию при применении предмета инвентаря;
- on{use,act,tak,inv,walk} - обработчик, реакция перед вызовом соответствующих обработчиков, может запрещать цепочку;
- after{use,act,tak,inv,walk} - обработчик, реакция после действий игрока.

- :step() -- такт игры;
- :lastdisp([v]) -- установить/взять последний вывод;
- :display(state) -- отобразить вывод;
- :lastreact([v]) -- установить/взять последнюю реакцию;
- :reaction([v]) -- установить/взять текущую реакцию;
- :events(pre, bg) -- установить/взять события живых объектов;
- :cmd(cmd) -- выполнение команды INSTEAD;

Игрок (player)

Кроме методов obj, добавлены следующие методы:

- :moved() -- игрок сделал перемещение в текущем такте игры;
- :need_scene([v]) -- нужна отрисовка сцены в данном такте;
- :inspect(w) -- найти объект (видимый) в текущей сцене или себе самом;
- :have(w) -- поиск в инвентаре;
- :useit(w) -- использовать предмет;
- :useon(w, ww) -- использовать предмет на предмет;
- :call(m, ...) -- вызов метода игрока;
- :action(w) -- действие на предмет (act);
- :inventory() -- вернуть инвентарь (список, по умолчанию это obj);
- :take(w) -- взять объект;
- :walk/walkin/walkout -- переходы;
- :go(w) -- команда идти (проверяет доступность переходов);