Справочное пособие по INSTEAD

Александр Яковлев
oreolek@jabber.ru

при участии Петра Косых
gl00my@jabber.ru

September 1, 2012

Contents

1 Введение

Данный справочник написан в предположении, что читатель знаком с основами объектно-ориентированног программирования.

Игры для движка STEAD пишутся на языке Lua. Знание этого языка будет очень полезным при написании игр, но я постараюсь сделать описание настолько подробным, что даже новичок в этом языке смог создавать игры для INSTEAD без проблем. Между прочим, знающим Lua будет небезынтересно посмотреть код движка.

Figure 1: INSTEAD с запущенной игрой

Главное окно игры содержит информацию о статической и динамической части сцены, активные события и картинку сцены с возможными переходами в другие сцены (в графическом интерпретаторе).

Динамическая часть сцены составлена из описаний объектов сцены, она отображается всегда. Она может выглядеть так: «Стоит стол. Рядом стоит стул». Если динамическая часть пуста, то игроку не с чем контактировать в сцене.

Статическая часть сцены описывает саму сцену, её «декорации». Она отображается при показе сцены (единожды или каждый раз – решает автор игры), или при повторении команды look (в графическом интерпретаторе при щелчке на названии сцены).

Игрок имеет собственный **инвентарь**. В нём лежат объекты, доступные на любой сцене. Чаще всего инвентарь рассматривают как некую «котомку», в которой лежат объекты; в этом случае каждый объект считают предметом. Такая трактовка практична, обыденна и интуитивна; но не единственна. Понятие инвентаря является условным, ведь это лишь контейнер. В нём могут находиться такие объекты, как «открыть», «потрогать», «лизнуть». Можно наполнить его объектами «нога», «рука», «мозг». Автор игры свободен в определении этих понятий, но он также должен определить действия игрока над ними.

На рисунке 1 очень чётко видны границы между этими областями. Главное окно имеет бежевый фон, инвентарь – чёрный. Динамическая часть идёт сразу после ссылок перехода и выделена курсивом; статическая часть отпечатана обычным шрифтом.

Действиями игрока могут быть:

- осмотр сцены
- действие на объект сцены
- действие на объект инвентаря
- действие объектом инвентаря на объект сцены
- действие объектом инвентаря на объект инвентаря

Осмотр сцены – это чаще всего неявное действие. Игрок входит в комнату, он автоматически осматривает её.

Действие на объект сцены обычно понимается как изучение объекта, или использование его. Например, если в сцене существует объект «чашка кофе», то действием на него может быть выпивание кофе, тщательный осмотр чашки, разбивание чашки или перемещение чашки в инвентарь. Это определяется только автором и никем другим.

Действие на объект инвентаря понимается аналогично. Например, если в инвентаре лежит объект «яблоко», его можно съесть или осмотреть. С другой стороны, если в инвентаре лежит объект «осмотреть», то действие над ним будет трудно описать логически.

Действие объектом инвентаря на объект сцены – это чаще всего использование или передача объекта. Например, действие объектом «нож» на объект «бармен» может означать передачу ножа бармену, угрозу ножом бармену, убийство ножом бармена и многое другое.

Действие объектом инвентаря на объект инвентаря понимается так же свободно. Это может быть соединение предметов («сарделька» + «кетчуп») в одно («сарделька с кетчупом»), либо использование («открыть» + «ящик»).

Эти примеры подробно показывают первую из идей STEAD – гибкость. Автор свободен в своей фантазии и может трактовать все понятия движка как хочет.

Игра представляет из себя каталог, в котором должен находиться скрипт main.lua. Другие ресурсы игры (скрипты на lua, графика и музыка) должны находиться в рамках этого каталога. Все ссылки на ресурсы делаются относительно текущего каталога – каталога игры.

Игра начинается именно с main.lua. В начале файла main.lua может быть определён заголовок, состоящий из тегов. Теги должны начинаться с символов комментария `--`. На данный момент существуют два тега: `$Name:`, который должен содержать название игры, и `$Version:`,который указывает версию игры. Пример использования тега:

```
-- $Name: Самая интересная игра!$
```

После тегов следует указать версию интерпретатора, которую требует игра:

```
instead_version "1.3.0"
```

Если это указание отсутствует, то STEAD будет работать в режиме совместимости, используя устаревшее API. Описание устаревшего API находится в предыдущих версиях данного справочника и убрано из данного издания для краткости.

Интерпретатор ищет доступные игры в следующих каталогах:

Unix версия интерпретатора просматривает `/usr/local/share/instead/games` (по умолчанию), а также `~/.instead/games`.

Windows сборка использует каталог `куда-вы-установили-INSTEAD\games`.

Расположение пользовательских игр также зависит от вашей версии Windows. Например, в Windows Vista игры могут лежать в каталоге `AppData\Local\instead\games`.

На данный момент активно развивается только графическая ветка интерпретатора – INSTEAD-SDL. Поэтому справочник в бóльшей мере описывает её возможности.

2 Сцена

Сцена – это единица игры, в рамках которой игрок может изучать все объекты сцены и взаимодействовать с ними. В игре должна быть сцена с именем main.

```
main = room {
        nam = 'главная комната',
        dsc = 'Вы в большой комнате.',
};
```

Отмечу, что пример выше является минимальной игрой для INSTEAD. Это некий «Hello, World», который я рекомендую сохранить под именем main.lua и поместить в отдельную папку в каталоге для игр.

Атрибут `nam` (имя) является необходимым для любого объекта. Для сцены это – то, что будет заголовком сцены при её отображении. Имя сцены также используется для её идентификации при переходах.

Атрибут `dsc` – это описание статической части сцены, которое выводится при входе в сцену или выполнении команды look.

Внимание!!! Если для вашего творческого замысла необходимо, чтобы описание статической части сцены выводилось на каждом ходу (а не только при первом входе в сцену), вы можете определить для своей игры параметр forcedsc (в начале игры).

```
game.forcedsc = true;
```

Или, аналогично, задать атрибут forcedsc для конкретных сцен.

Для длинных описаний удобно использовать запись вида:

```
dsc = [[ Очень длинное описание... ]],
```

При этом переводы строк игнорируются. Если вы хотите, чтобы в выводе описания сцены присутствовали абзацы – используйте символ `^`.

```
dsc = [[ Первый абзац. ^^
Второй Абзац.^^

Третий абзац.^
На новой строке.]],
```

Символы `^` и `|` можно экранировать в строках, если вы хотите их напечатать для пользователя:

```
p " \\^ \\| "
p [[ \^ \| ]]
```

К текущей сцене можно обратиться через функцию `here()`.

3 Объект

3.1 Нормальные объекты

Объекты – это единицы сцены, с которыми взаимодействует игрок.

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

Имя объекта `nam` используется для адресации объекта.

`dsc` – описатель объекта. Он будет выведен в динамической части сцены. Фигурными скобками отображается фрагмент текста, который будет являться ссылкой в графическом интерпретаторе. Если вы забудете сделать ссылку, то интерпретатор не выдаст ошибки, но игроки не смогут взаимодействовать с объектом.

`act` – это обработчик, который вызывается при действии пользователя на объект сцены. Если объект находится в инвентаре, то действие с ним будет передаваться другому обработчику – `inv`.

До сих пор в примерах приводились примитивные обработчики, которые всего лишь возвращают определённую строку. В примере выше обращение к объекту вызовет банальную реакцию: интерпретатор напечатает строку «Гм... Просто стол...». Хуже того: он будет отвечать тем же образом каждый раз при обращении к объекту. Это не совсем гибкий подход, поэтому STEAD позволяет определить любой атрибут объекта как функцию. Так, возможно построить такую конструкцию:

```
apple = obj {
        nam = 'яблоко',
        dsc = function(s)
                if not s._seen then
                        return 'На столе {что-то} лежит.';
                else
                        return 'На столе лежит {яблоко}.';
                end
        end,
        act = function(s)
                if s._seen then return 'Это яблоко!';
                else
                        s._seen = true;
                        return 'Я присматриваюсь и понимаю, что это -- яблоко.!';
                end
        end,
};
```

Если атрибут или обработчик оформлен как функция, то обычно первый аргумент функции `(s)` есть сам объект. В данном примере, при показе сцены будет в динамической части сцены будет текст: «На столе что-то лежит». При взаимодействии со ссылкой «что-то», переменная `_seen` объекта `apple` будет установлена в `true`, и мы увидим, что это было яблоко.

Запись `s._seen` означает, что переменная `_seen` размещена в объекте `s` (то есть, `apple`). В языке Lua переменные необязательно объявлять заранее, при первом обращении к ней переменная `apple._seen` появится сама; но хорошим тоном будет заранее **проинициализировать** переменную со значением `false`.

Подчёркивание в имени переменной означает, что она **попадёт** в файл сохранения игры. Сохраняются все переменные, название которых начинается с с подчёркивания.

Вы можете переопределить функцию `isForSave(k)`, если вас это не устраивает, либо (рекомендованный вариант) воспользоваться блоком `var`, в котором сохраняются все переменные. Для краткости примеры этого руководства не следят за сохранением переменных.

Внимание!!! Переменные в любом случае не записываются в файл сохранения, если они не размещены в одном из перечисленных типов объектов: комната, объект, диалог, игра, игрок, глобальное пространство.

В файл сохранения могут быть записаны строки, числа, логические значения, ссылки на объекты и конструкции `code` (о них чуть ниже).

Также вы можете определять переменные при помощи блоков `var` и `global`, о которых будет рассказано позже.

Рассмотрим другой пример:

```
button = obj {
  nam = "кнопка",
  dsc = "На стене комнаты видна большая красная {кнопка}.",
  act = code[[ p 'Я нажал на кнопку.' ]]
}
```

Здесь я использовал две новых конструкции: `code` и `p`. Конструкция `code [[ код ]]` - это короткая форма записи для:

```
act = function(self, ...)
 код
end,
```

При этом внутри `code` определены переменные `self` – текущий объект, `arg1`, `arg2`, ..., `arg9` – параметры функции, и массив параметров `args[]`.

Также для сокращения записи нужна функция `p`. Она возвращает текст, заканчивая его пробелом. Всего определено три подобных функции:

p – выводит переданный текст, заканчивая его пробелом

pn – выводит переданный параграф, заканчивая его переводом строки

pr – выводит переданное как есть

Lua позволяет опускать скобки вокруг параметров, если параметр всего один.

Если необходимо показать, что действие невыполнимо, можно вернуть из обработчика значение `false` или `nil`. При этом будет отображено описание по умолчанию – так, для обработчика `act` это будет `game.act`.

Если обработчик не возвращает ничего, то он выполняется, а отображается описание по умолчанию.

3.2 Облегчённые объекты

Иногда, сцену нужно наполнить декорациями, которые обладают ограниченной функциональностью, но делают игру разнообразней. Для этого можно использовать облегчённый объект. Например:

```
sside = room {
        nam = 'южная сторона',
        dsc = [[Я нахожусь у южной стены здания института. ]],
        act = function(s, w)
                if w == "подъезд" then
                        ways():add('stolcorridor');
                        return "Я подошёл к подъезду. Хм -- зайти внутрь?";
                end
        end,
        obj = {vobj("подъезд", "Я вижу небольшой {подъезд}.")},
};
```

Как видим, `vobj` позволяет сделать лёгкую версию статического объекта, с которым тем не менее можно взаимодействовать (за счёт определения обработчика `act` в сцене и анализа ключа объекта). `vobj` также вызывает метод `used`, при этом в качестве третьего параметра передаётся объект, воздействующий на виртуальный объект.

Синтаксис `vobj` таков: `vobj(имя, описатель)`.

Существует модификация объекта `vobj` под именем `vway`. `vway` реализует ссылку. Синтаксис и пример:

```
vway(имя, описание, сцена назначения);
obj = { vway("дальше", "Нажмите {здесь}.", 'nextroom') }
```

Вы можете динамически заполнять сцену объектами `vobj` или `vway` с помощью методов `add` и `del`.

В довершение, определена также упрощённая сцена `vroom`. Синтаксис:

```
vroom(имя перехода, сцена назначения)
```

Ниже приводится несколько примеров и трюков с подобными объектами:

```
home.objs:add(vway("next", "{Дальше}.", 'next_room');
home.objs:del("next");
home.objs:add(vroom("идти на запад", 'mountains');
if not home.obj:srch('Дорога') then
 home.obj:add(vway('Дорога', 'Я заметил {дорогу} в лес.', 'forest'));
end
obj = {vway('Дорога', 'Я заметил {дорогу} в лес.', 'forest'):disable()},
objs()[1]:disable();
objs()[1]:enable();
```

3.3 Динамическое создание объектов

Вы можете использовать аллокаторы `new` и `delete` для создания и удаление динамических объектов:

```
new("obj { nam = 'a' ..... }")
put(new [[obj {nam = 'test' } ]]);
put(new('myconstructor()');
```

Созданный объект будет попадать в файл сохранения. `new()` возвращает реальный объект; чтобы получить его имя, если это нужно, используйте функцию `deref()`.

4 Некоторые манипуляции с объектами

4.1 Объект и сцена

Ссылкой на объект называется текстовая строка, содержащая имя объекта при его создании.

Например: `'table'` – ссылка на объект `table`.

Для того, чтобы поместить в сцену объекты, нужно определить массив `obj`, состоящий из ссылок на объекты:

```
main = room {
        nam = 'главная комната',
        dsc = 'Вы в большой комнате.',
        obj = { 'tabl' },
};
```

4.2 Объекты, связанные с объектами

Объекты тоже могут содержать атрибут `obj`. При этом, список будет последовательно разворачиваться. Например, поместим на стол яблоко:

```
apple = obj {
        nam = 'яблоко',
        dsc = 'На столе лежит {яблоко}.',
};

table = obj {
        nam = 'стол',
        dsc = 'В комнате стоит {стол}.',
        obj = { 'apple' },
};
```

При этом, в описании сцены мы увидим описание объектов «стол» и «яблоко», так как `apple` – связанный с `table` объект.

4.3 Действия объектов друг на друга

Игрок может действовать объектом инвентаря на другие объекты. При этом вызывается обработчик `use` у объекта которым действуют и `used` – на которого действуют.

Например:

```
knife = obj {
        nam = 'нож',
        dsc = 'На столе лежит {нож}',
        inv = 'Острый!',
        tak = 'Я взял нож!',
        use = 'Вы пытаетесь использовать нож.',
};

tabl = obj {
        nam = 'стол',
        dsc = 'В комнате стоит {стол}.',
        act = 'Гм... Просто стол...',
        obj = { 'apple', 'knife' },
        used = 'Вы пытаетесь сделать что-то со столом...',
};
```

Если игрок возьмёт нож и использует его на стол, то увидит текст обработчиков `knife.use` и `tabl.used`.

`use` и `used` могут быть функциями. Тогда первый параметр это сам объект, а второй – ссылка на объект, на который направлено действие в случае `use` и объект, которым действие осуществляется в случае `used`.

`use` может вернуть статус `false`, в этом случае обработчик `used` не вызывается (если он вообще был). Статус обработчика `used` – игнорируется. Это будет выглядеть как:

```
return 'Строка реакции', false;
```

Возможно также действовать объектами сцены на объекты сцены; для этого нужно установить переменную `game.scene_use = true` или поставить `scene.use=true` в нужной комнате. В этом случае использование объектов сцены будет аналогично использованию объектов инвентаря.

4.4 Скрытие объектов

При помощи методов `enable` и `disable` становится возможным управлять появлением и исчезновением объектов.

Скрытый объект – это объект, который находится в сцене, но на данный момент словно бы «выключен». Он присутствует для движка, но не существует для игрока. Его описание не выводится и с ним невозможно контактировать. Это можно использовать, например, вместо динамического создания объектов.

Чтобы создать заведомо выключенный объект, необходимо воспользоваться конструкцией вида:

```
knife = {<...>}:disable()
```

Объект `knife` будет создан и тут же выключен.

Методы `enable()` и `disable()` возвращают сам объект. Они присутствуют у любого объекта и списка объектов.

5 Смена сцен

Как только главный герой уходит со сцены, декорации меняются. Но чтобы игрок ушёл из нашей сцены, он должен знать, куда идти.

Для перехода между сценами используется атрибут сцены – список `way`.

```
room2 = room {
        nam = 'зал',
        dsc = 'Вы в огромном зале.',
        way = { 'main' },
};

main = room {
        nam = 'главная комната',
        dsc = 'Вы в большой комнате.',
        obj = { 'tabl' },
        way = { 'room2' },
};
```

При этом, вы сможете переходить между сценами `main` и `room2`. Как вы помните, `nam` может быть функцией, и вы можете генерировать имена сцен на лету, например, если вы хотите, чтобы игрок не знал название сцены, пока не попал на неё.

При переходе между сценами движок вызывает обработчик `exit` из текущей сцены и `enter` той сцены, куда идёт игрок. `exit` и `enter` могут быть функциями – тогда первый параметр это сам объект, а второй – ссылка на комнату куда игрок хочет идти (для `exit`) или из которой уходит (для `enter`).

После перехода вызываются обработчики `left` прошлой сцены и `entered` сцены, куда перешёл игрок.

Например:

```
room2 = room {
        enter = function(s, f)
                if f == main then
                        return 'Вы пришли из комнаты.';
                end
        end,
        nam = 'зал',
        dsc = 'Вы в огромном зале.',
        way = { 'main' },
        exit = function(s, t)
                if t == main then
                        return 'Я не хочу назад!', false
                end
        end,
};
```

Как видим, обработчики могут возвращать два значения: строку и статус. В нашем примере функция `exit` вернёт `false`, если игрок попытается уйти из зала в `main`. `false` блокирует переход. Такая же логика работает и для `enter`. Кроме того, она работает и для обработчика `tak` (о нём чуть позже).

Если требуется перейти на другую сцену автоматически, можно использовать функцию `walk` со ссылкой на сцену как параметром:

```
return walk('main');
goto('main');
```

6 Специальные типы объектов

6.1 Инвентарь

Инвентарь проще всего возвращается функцией inv(). Он представлен списком, поэтому для него справедливы все их трюки (см. соответствующий раздел).

Простейший вариант сделать объект, который можно брать – определить у него обработчик tak.

Если предмет сцены имеет обработчик `tak` и НЕ имеет обработчика `act` [1], то при действии на нём вызывается не `act`, а `tak`; после этого предмет перемещается в инвентарь. Это происходит вот так:

```
apple = obj {
        nam = 'яблоко',
        dsc = 'На столе лежит {яблоко}.',
        tak = 'Вы взяли яблоко.',
};
```

6.2 Игрок

Игрок в STEAD представлен объектом `pl`. Тип объекта – `player`.

Атрибут `obj` представляет собой инвентарь игрока.

Объекты игрока можно воспринимать как объекты инвентаря, в этом случае верны следующие конструкции:

```
remove('knife', me());
put('knife', me());
take('knife', me());
where('knife');
```

Необходимо отметить, что это верно не в каждой игре.

6.3 Игра

Игра представлена объектом `game`. Он хранит в себе указатель на текущего игрока (`'pl'`) и некоторые параметры. Например, вы можете указать в начале своей игры кодировку текста следующим образом:

```
game.codepage="UTF-8";
```

Кроме того, объект `game` может содержать обработчики по умолчанию `act`, `inv`, `use`, которые будут вызваны, если в результате действий пользователя не будут найдены никакие другие обработчики. Например, вы можете написать в начале игры:

```
game.act = 'Не получается.';
game.inv = 'Гм.. Странная штука..';
game.use = 'Не сработает...';
```

На практике полезно что-то вроде:

```
game.inv = function()
 local reaction = {
  [1] = 'Либо я ошибся карманом, либо мне нужно что-то другое.',
  [2] = 'Откуда у меня в кармане ЭТО?!',
```

[1] Пользуясь случаем: я считаю, что tak – немного неудобное имя. Именно так. – А.Я.

```
  [3] = 'Сам не понял, что достал. Положу обратно.',
  [4] = 'Это что-то неправильное.',
  [5] = 'В моих карманах что только не залёживается...',
  [6] = 'Я не представляю, как я могу тащить ЭТО с собою.',
  [7] = 'Мне показалось или оно на меня смотрит?',
 };
 return reaction[rnd(#reaction)];
end;
```

6.4 Таймер

Таймер – это объект `timer`, который служит для отсчёта **реального** времени (в миллисекундах). В этом его существенное отличие от атрибутов `life`, которые служат для измерения игрового времени (в шагах).

Для управления таймером используются функции:

timer:set(ms) задать интервал таймера в миллисекундах

timer:stop() отключить таймер

timer.callback(s) функция-обработчик таймера, которая вызывается через заданный интервал времени

Двоеточия при вызове `set` и `stop` важны, их не стоит заменять на точки. По умолчанию таймер выключён, и не имеет заданного обработчика. Если таймер включён, то обработчик вызывается с заданным интервалом.

Чтобы включить таймер после выполнения `stop`, достаточно переинициализировать его командой `set` с прежним интервалом.

Пример использования таймера:

```
timer.callback = function(s)
 main.time = main.time + 1;
 return "look";
end
timer:set(100);

main = room {
 time = 1,
 forcedsc = true,
 nam = 'Таймер',
 dsc = function(s)
  return 'Демонстрация: '..tostring(s._time);
 end
};
```

6.5 Ввод с клавиатуры

Ввод с клавиатуры анализируется при помощи объекта `input`. Этот объект принимает все нажатия клавиш. По умолчанию он не имеет запрограммированных обработчиков, поэтому ничего не выполняет.

Создание нового обработчика для клавиши выполняется командой `input.key(s, pressed, key)`, где `pressed` – нажатие или отжатие, `key` – символьное имя клавиши;

Если обработчик возвращает не `nil`, то клавиша обрабатывается **им**, а не интерпретатором. Таким образом, можно переопределить обработку любой клавиши.

Например:

```
input.key = function(s, pr, key)
 if not pr or key == "escape" then return
 elseif key == 'space' then key = ' '
 elseif key == 'return' then key = '^';
 end
 if key:len() > 1 then return end
 main.txt = main.txt:gsub('_$','');
 main.txt = main.txt..key..'_';
 return "look";
end

main = room {
 txt = '_',
 forcedsc = true,
 nam = 'Клавиатура',
 dsc = function(s)
  return 'Демонстрация: '..tostring(s._txt);
 end
};
```

7 Диалоги

Третьим важным типом в движке являются диалоги. Диалоги – это особый подвид сцен, содержащий только фразы. Например, диалог может выглядеть следующим образом:

```
povardlg = dlg {
 nam = 'на кухне',
 dsc = 'Передо мной полное лицо повара...',
 obj = {
  [1] = phr('Мне вот-этих зелёненьких... Ага - и бобов!', 'На здоровье!'),
  [2] = phr('Картошку с салом, пожалуйста!', 'Приятного аппетита!'),
  [3] = phr('Две порции чесночного супа!!!', 'Прекрасный выбор!'),
  [4] = phr('Мне что-нибудь лёгонькое, у меня язва...', 'Овсянка!'),
 },
};
```

`phr` – создание фразы. Фраза содержит вопрос, ответ и реакцию (реакция в данном примере отсутствует). Когда игрок выбирает одну из фраз, фраза отключается. Когда все фразы отключатся диалог заканчивается. Реакция – это строка кода на lua который выполнится после отключения фразы.

`_phr` – создание выключенной фразы. Она не видна изначально, но её можно включить с помощью функции `pon()` (см. ниже). По смыслу это эквивалентно `phr(<...>):disable()`.

Вот как создаётся фраза:

```
[1] = phr('Можно задать вопрос?', 'Конечно!', [[pon(1);]]),
```

В реакции может быть любой lua код (простейшей реакцией является возвращение строки), но в STEAD определены наиболее часто используемые функции:

pon(n...) включить фразы диалога с номерами n... (в нашем примере – чтобы игрок мог повторно взять еду того-же вида).

poff(n...) выключить фразы диалога с номерами n...

prem(n...) удалить (заблокировать) фразы диалога с номерами n... Удаление означает невозможность включения фраз. `pon(n..)` не приведёт к включению фраз.

pseen(n...) вернет true, если все заданные фразы диалога видимы.

punseen(n...) вернет true, если все заданные фразы диалога невидимы.

Если параметр `n` не указан, действие относится к текущей фразе.

Как ответ, так и реакция могут быть функциями. Вы можете закончить диалог, выполнив в реакции функцию `back()`.

Переход в диалог осуществляется как переход на сцену:

```
return walk('povardlg');
```

Вы можете переходить из одного диалога в другой диалог – организовывая иерархические диалоги.

Также, вы можете прятать некоторые фразы при инициализации диалога и показывать их при некоторых условиях.

Вы можете включать/выключать фразы не только текущего, но и произвольного диалога, с помощью методов объекта диалог `pon/poff`. Например:

```
shopman:pon(5);
```

Если номер фразы не указан, то это означает, что действие относится к текущей фразе:

```
phr('a', 'b', [[ pon() ]]);
```

У самих диалогов также есть метод `empty`. С его помощью можно проверять, пуст ли диалог:

```
if dlg:empty() then p "Диалог пуст" end
```

8 О списках

Каждый атрибут-список имеет методы:

add Добавление элемента в список. Необязательным вторым параметром является позиция в списке.

del Удаление элемента из списка. Выключённые элементы не удаляются.

purge Удаление элемента из списка. Выключённые элементы также удаляются.

look Получение индекса элемента в списке по идентификатору

srch Проверка на наличие объекта в списке по его nam. Возвращает 2 значения: идентификатор и позицию; если объекта нет, вернёт nil. Например:

```
objs():srch('Ножик')
```

replace Замена объекта в списке другим

set Изменение объекта по номеру. Например, этот пример присвоит заменит первый объект списка:

```
objs():set('knife',1);
```

disable Скрытие объекта в списке; отличается от удаления тем, что может быть возвращён к жизни через метод enable

enable Показ скрытого объекта

zap Обнуление списка

cat(b) Склеивает список со списком b

cat(b,pos) Добавляет в список список b на позицию pos

disable_all Аналогично disable, но массово

enable_all Аналогично enable, но массово

Параметрами методов могут быть объекты, их идентификаторы и имена.

9 Функции

Замечание. Если аргумент у функции единственен и текстовый, то скобки вокруг аргумента можно опускать. Большинство описываемых в этом разделе функций с несколькими параметрами могут использоваться без указания всех параметров. В этом случае опущенные параметры принимают значение по умолчанию (обычно – текущая сцена). .

inv() возвращает список инвентаря

objs() возвращает список объектов указанной сцены; если сцена не указана, то возвращает список объектов текущей сцены.

ways() возвращает список возможных переходов из указанной сцены; если сцена не указана, то возвращает список возможных переходов из текущей сцены.

me() возвращает объект pl (объект игрока)

here() возвращает текущую сцену

where() возвращает сцену в которой помещен объект (если он был добавлен с использованием функций put, drop, move)

from() возвращает прошлую сцену

ref(nam) возвращает указанный объект:

```
ref('home') == home
```

nam может быть функцией, строкой или текстовой ссылкой

deref(object) возвращает ссылку строкой для объекта:

```
deref(knife) == 'knife'
```

nameof(object) возвращает `nam` объекта

have(object) проверяет, есть ли объект в инвентаре по имени объекта или по его nam

move(from, where, to) переносит объект из текущей сцены в другую

movef(from, where, to) действует так же, но добавляет объект в начало списка

moveto(from,where,to,position) перемещает элемент в нужную позицию списка объектов

seen(object,scene) проверяет, есть ли объект в указанной сцене (в текущей, если сцена не указана)

exist(object,scene) делает то же самое, что и `seen`, но учитывает и выключенные объекты

path(scene,scene) проверяет, есть ли путь в нужную сцену из указанной

drop(object,scene) выбрасывает объект из инвентаря в указанную сцену (в текущую, если сцена не указана)

dropf(object,scene) то же, что drop, но объект появляется в начале списка

dropto(object,scene,position) выбрасывает объект на нужную позицию

put(object,scene) кладёт предмет в указанную сцену; в текущую, если сцена не указана

putf(object,scene) кладёт предмет в начало списка

putto(object,scene,position) кладёт предмет в указанную позицию списка

remove(object,scene) удаляет предмет из указанной сцены; из текущей, если сцена не указана

take(object,scene) перемещает объект из указанной(текущей) сцены в инвентарь

takef(object) берёт объект в инвентарь на первую позицию

taketo(object,position) берёт объект в инвентарь на указанную позицию

taken(object) проверяет, взят ли уже объект

rnd(m) возвращает случайное целое значение от 1 до m

walk(destination) переносит в сцену w; используется в конструкции вида `return walk('inmycar');`

change_pl(player) переключает на другого игрока (со своим инвентарём и позицией), также используется в
`return`. Может использоваться для переключения между разными инвентарями.

back() переносит в предыдущую сцену (аналогично walk)

time() возвращает текущее время игры в активных действиях.

cat(...) возвращает строку – склейку строк-аргументов. Если первый аргумент nil, то функция возвращает nil.[2]

par(...) возвращает строку – склейку строк-аргументов, разбитых строкой-первым параметром.

Следующие записи эквивалентны:

```
ref('home').obj:add('chair');
home.obj:add('chair');
objs('home'):add('chair');
put('chair', 'home');
```

[2]Функция легко и просто заменяется обычным оператором склейки строк Lua: `строка1 .. строка2`. Тем не менее, этот оператор выдаст ошибку при склейке nil

Если вы хотите перенести объект из произвольной сцены, вам придётся удалить его из старой сцены с помощью метода del. Для создания сложно перемещающихся объектов, вам придётся написать свой метод, который будет сохранять текущую позицию объекта в самом объекте и делать удаление объекта из старой сцены. Вы можете указать исходную позицию (комнату) объекта в качестве третьего параметра move.

```
move('mycat','inmycar', 'forest');
```

Отдельно следует упомянуть функции альтернативного вывода текста. Их назначение можно понять по следующим формам записи:

```
act = function(s)
       p "Я осмотрел его...';
       p "Гммм...."
end

act = function(s)
       return "Я осмотрел его... Гммм...."
end
```

p Добавляет строку и пробел в буфер

pn Добавляет строку и перевод строки в буфер

pclr Очистка буфера

pget Получение содержимое буфера на текущий момент

Другой пример:

```
life = function(s)
     p 'Ветер дует мне в спину.'
     return pget(), true
end
```

10 Добавление динамики в игру

Игра измеряет время в своих единицах – в шагах, или активных действиях. Каждое действие игрока – это его шаг, пусть даже он тратит его на то, чтобы ещё раз осмотреться. Что он увидит нового? Что изменится в мире игры за это время?

Именно для того, чтобы задать динамику мира, существует система с говорящим названием life.

Вы можете определять обработчики, которые выполняются каждый раз, когда время игры увеличивается на 1. Например:

```
mycat = obj {
        nam = 'Барсик',
        lf = {
                [1] = 'Барсик шевелится у меня за пазухой.',
                [2] = 'Барсик выглядывает из за пазухи.',
                [3] = 'Барсик мурлычет у меня за пазухой.',
                [4] = 'Барсик дрожит у меня за пазухой.',
                [5] = 'Я чувствую тепло Барсика у себя за пазухой.',
```

```
	},
	life = function(s)
		local r = rnd(5);
		if r > 2 then
			return;
		end
		r = rnd(5);
		return s.lf[r];
	end,
```

В этом примере кот по имени Барсик, сидя в инвентаре у игрока, будет на каждом шагу показывать свою активность. Но приведённый код пока что не будет работать. Для того, чтобы объявить объект «живым», следует добавить его в соответствующий список с помощью функции `lifeon()`.

```
inv():add('mycat');
lifeon('mycat');
```

Любой объект или сцена могут иметь свой обработчик `life`, который вызывается каждый раз при смене текущего времени игры, если объект или сцена были добавлены в список живых объектов с помощью `lifeon`. Не забывайте удалять живые объекты из списка с помощью `lifeoff`, когда они больше не нужны. Это можно сделать, например, в обработчике `exit`, или любым другим способом.

Вы можете вернуть из обработчика life второй код возврата, важность. (true или false). Если он равен true, то возвращаемое значение будет выведено ДО описания объектов; по умолчанию значение равно false.

Если вы хотите «очистить экран», то можно воспользоваться следующим хаком. Из метода `life` доступна переменная `ACTION_TEXT` – это тот текст, который содержит реакцию на действие игрока. Соответственно, сделав `ACTION_TEXT = nil`, можно «запретить» вывод реакции. Например, для перехода на конец игры можно сделать:

```
ACTION_TEXT = nil
return walk('theend'), true
```

Также для добавления динамики в игру служат объекты timer и input, см. разделы 6.4 и 6.5.

11 Краски и звуки

В чём очевидное преимущество графического интерпретатора над текстовой веткой – это то, что он может говорить и показывать. Проще говоря, вы можете добавить в игру графику и музыку.

Графический интерпретатор анализирует атрибут сцены `pic`, и воспринимает его как путь к картинке-иллюстрации для сцены. Если в текущей сцене не определён атрибут `pic`, то берётся `game.pic`. Если не определён и он, то картинка не отображается.

Unix-версия движка поставляется в исходных кодах, поэтому поддержка форматов графики и музыки в основном определяется возможностями библиотеки SDL, установленной в вашей системе. В большинстве случаев обеспечивается поддержка всех основных форматов. Для графики я рекомендую использовать форматы PNG, JPG и GIF (также поддерживаются анимированные GIF). Новые версии движка поддерживают GIF-анимацию. Те же правила действуют для музыки. Здесь строгих рамок нет, поэтому старайтесь не использовать редких форматов. Проверена поддержка WAV, MP3, OGG, FLAC, XM, MOD, IT, S3M.

Вы также можете встраивать графические изображения в текст или в инвентарь с помощью функции `img`. Например:

```
knife = obj {
	nam = 'Нож'..img('img/knife.png'),
}
```

Чтобы разделить иллюстрацию и описание предмета, можно задать иллюстрацию в атрибуте `disp`:

```
knife = obj {
 nam = 'Нож';
 disp = img('img/knife.png'),
}
```

Тогда игра будет обращаться к предмету как `'нож'`, но игроку каждый раз вместо названия предмета будет показываться его иллюстрация.

При этом изображения можно комбинировать, накладывая друг на друга. Очень удобны в этом отношении PNG и GIF, потому что они позволяют задать прозрачность фона. В случае анимированных GIF наложение происходит только с первым кадром. Это выглядит следующим образом:

```
pic = "gfx/mycat.png;gfx/milk.png@100,100;gfx/fish.png@c20,20"
```

Картинки складываются стопкой снизу вверх. Самая первая картинка является фоном. Она должна быть самой большой; её границы - это и границы всей картинки. Точка с запятой разделяет изображения; после неё идёт сначала путь к картинке, затем символ `@` и координаты точки фона, куда будет помещён левый верхний угол накладываемой картинки. Координаты считаются от левого верхнего угла фонового изображения. Если после символа `@` идёт буква `c`, то в эту точку будет помещён не левый верхний угол, а **центр** накладываемой картинки.

Для того, чтобы вставить картинку у определённого края, нужно вызвать функцию `imgl()` или `imgr()`. Такие картинки не могут быть ссылками.

Также можно использовать псевдофайлы:

Псевдофайл `pad` задаёт отступы в изображении:

`imgl 'pad:16,picture.png'` – отступы по 16 пикселов от каждого края

`imgl 'pad:0 16 16 4,picture.png'` – вверху 0, справа 16, внизу 16, слева 4

`imgl 'pad:0 16,picture.png'` – вверху 0, справа 16, внизу 0, слева 16

Псевдофайл `blank` рисует пустое изображение:

```
dsc = img 'blank:32x32'..[[Строка с пустым изображением.]];
```

Псевдофайл box рисует квадрат. Вы задаёте его размер, цвет и прозрачность (от 0 до 255):

```
dsc = img 'box:32x32,red,128'..[[Красный полупрозрачный квадрат.]];
```

Теперь о звуке. Фоновая музыка задаётся с помощью функции:

```
set_music(имя музыкального файла, количество проигрываний)
```

Она проигрывается циклически бесконечно, если количество проигрываний не задано.

`get_music()` возвращает текущее имя трека.

Функция `get_music_loop` возвращает текущий счетчик проигрываний мелодии. 0 – означает вечный цикл. n – количество оставшихся проигрываний. -1 – проигрывание текущего трека закончено.

Помимо фонового сопровождения, `set_sound()` позволяет проиграть звуковой файл. `get_sound()` возвращает имя звукового файла, который будет проигран.

12 Модули

Начиная с версии 1.2.0, с INSTEAD поставляются различные дополнительные модули, которые просто подключать в игре:

```
--$Name: Моя игра!$
require "para"
require "dbg"
```

Информацию об использовании модулей смотрите на вики INSTEAD.

13 Трюки

13.1 Переопределение стандартных функций и объектов

Переопределять стандартные функции и объекты INSTEAD удобно при помощи функций `hook` и `inherit`.

```
input.key = hook(input.key,
function(f, s, down, key, ...)
        if down and key == 'f7' then return 'look' end
        return f(s, down, key, unpack(arg))
end)
```

```
obj = inherit(obj,function(v)
    v.act = hook(v.act, function(f, s, ...)
        local r = f(s, unpack(arg))
        s._seen = true
        return r
    end)
    return v
end)
```

13.2 Инициализация игры

Вместо того, чтобы инициализировывать игру в комнате `main`, можно воспользоваться функцией `init`:

```
function init()
    prefs.counter  = 0
    put 'fork'
    lifeon 'dog'
end
```

13.3 Переменные STEAD

В версии 1.2.0 был представлен следующий подход к созданию переменных:

```
var {
        i = "a";
        z = "b";
    };
```

Переменные, объявленные таким образом, попадают в файл сохранения независимо от того, с какого символа начинаются их имена.

Для того, чтобы задавать переменные таким образом, необходимо указать `instead_version` не менее 1.2.0.

Переменная в vars может содержать участок кода (`code [[...]]`), который также будет сохранён. Таким образом, вы можете создавать сложные сохраняемые обработчики.

Также возможно определять глобальные переменные:

```
global {
    global_var = 1;
}
```

Глобальная переменная доступна на протяжении всей игры.

13.4 Форматирование

SDL-INSTEAD поддерживает простое форматирование текста с помощью функций:

txtc(текст) – разместить текст по центру

txtr(текст) – разместить текст справа

txtl(текст) – разместить слева

txtb(текст) – полужирное начертание

txtem(текст) – начертание курсивом

txtu(текст) – подчёркнутый текст

txtst(текст) – зачёркнутый текст

txtnb(текст) – делает все пробелы в тексте неразрывными.

Немного по последней функции. Код `txtnb(' ')` даст вам один неразрывный пробел. Неразрывный пробел - это пробел, который нельзя заменить переходом на новую строку. То есть, если вы напишете `txtnb('a b')`, то это будет гарантировать вам, что `a` и `b` находятся на одной строке, а не на разных.

Также в версии 1.3.0 появилась функция txttab. С её помощью возможно создание таблиц. Её синтаксис:

```
txttab ("позиция", [[выравнивание]])
```

Здесь «позиция» выражается в пикселях либо процентах, а выравнивание может иметь значения `"left"`,
`"right"` и `"center"`:

```
txttab("90%","right").."слово"
txttab("50%").."слово"
"название"...txttab"60%".."цена^".."название2"..txttab"60%".."новая цена"
"название".. txttab("100%","right")..txtnb("два или три слова")
```

Значения `"left"`, `"right"` и `"center"` указывают точку, которая будет использоваться для позиционирова

left – начало слова

center – центр слова

right – правый конец слова

Текст, обрамлённый в `txtnb()`, воспринимается как одно слово.

13.5 Проверка правописания

Проверка правописания готовой игры – это большая головная боль. У вас есть примерно 100 Кб кода, в которых находятся около 80 Кб текста. Любая программа проверки орфографии будет сильно ругаться на синтаксис Lua и мешать. Один из способов проверки – использовать редактор Emacs.

Для проверки нужно установить сам Emacs и поддержку Lua к нему (lua-mode); дальнейшие операции с редактором:

1. Открыть нужный файл
2. Если русские буквы выглядят кракозябрами – выбираете в меню Options – Set Font/FontSet... шрифт fixed
3. Tools – Spell Checking – Select Russian Dict
4. Tools – Spell Checking – Spell-Check Buffer
5. Пробел для того, чтобы пропустить слово; Латинская a, чтобы игнорировать слово вообще; i чтобы добавить слово в словарь пользователя; цифры, чтобы заменить слово на один предложенных вариантов.

Если вы впервые видите этот редактор, я настоятельно НЕ рекомендую нажимать что-нибудь и щёлкать на что-нибудь непонятное. Будет только непонятнее.

13.6 Меню

Вы можете делать меню в области инвентаря, определяя объекты с типом menu. При этом, обработчик меню будет вызван после клика мыши. Если обработчик не возвращает текст, то состояние игры не изменяется. Например, реализация кармана:

```
pocket = menu {
        State = false,
        nam = function(s)
                if s.State then
                        return txtu('карман');
                end
                return 'карман';
        end,
        gen = function(s)
                if s.State then
                        s:enable_all();
                else
                        s:disable_all();
                end
        end,
        menu = function(s)
                if s.State then
                        s.State = false;
                else
                        s.State = true;
                end
                s:gen();
        end,
```

```
};

knife = obj {
        nam = 'нож',
        inv = 'Это нож',
};

inv():add(pocket);
put(knife, pocket);
pocket:gen();

main = room {
        nam = 'test',
};
```

13.7 Статус

Ниже представлена реализация статуса игрока в виде текста, который появляется в инвентаре, но не может быть выбран.

```
status = stat {
       nam = function(s)
               return 'Статус!!!';
       end
};

inv():add('status');
```

13.8 Кодирование исходного кода

Если вы не хотите показывать исходный код своих игр, вы можете закодировать его с помощью команды

```
sdl-instead -encode <путь к файлу> [выходной путь]
```

и использовать его с помощью lua функции `doencfile(путь к файлу)`.

Важное замечание: Лучше не использовать компиляцию игр с помощью luac, так как luac создаёт платформозависимый код. Таким образом, вам придётся выдавать сразу две скомпилированных версии: для 32-битных и 64-битных машин[3]. Однако, компиляция игр может быть использована для поиска ошибок в коде.

13.9 Создание собственного плейлиста

Вы можете написать для игры свой проигрыватель музыки, создав его на основе живого объекта, например:

```
tracks = {"mus/astro2.mod", "mus/aws_chas.xm", "mus/dmageofd.xm"}
mplayer = obj {
        nam = 'плеер',
```

[3] А в будущем, возможно, SDL-INSTEAD будет поддерживать больше платформ, поэтому использовать предложенный метод будет ещё выгоднее

```
        life = function(s)
                local n = get_music();
                local v = get_music_loop();
                if not n or not v then
                        set_music(tracks[2], 1);
                elseif v == -1 then
                        local n = get_music();
                        while get_music() == n do
                                n = tracks[rnd(4)]
                        end
                        set_music(n, 1);
                end
        end,
};
lifeon('mplayer');
```

13.10 Отладка

Для того, чтобы во время ошибки увидеть стек вызовов функций lua, вы можете запустить

```
sdl-instead -debug
```

Вы можете отлаживать свою игру вообще без instead. Например, вы можете создать следующий файл game.lua:

```
dofile("/usr/share/games/stead/stead.lua"); -- путь к stead.lua
dofile("main.lua"); -- ваша игра
game:ini();
iface:shell();
```

И запустите игру в lua: `lua game.lua`. При этом игра будет работать в примитивном shell окружении. Полезные команды: ls, go, act, use.... Теоретически движок можно таким образом привязать даже к CGI окружению.

Также рекомендуется использовать модуль `dbg`. После подключения этого модуля в инвентаре появится объект "отладка". Нажав на него (либо на F7), вы войдёте в диалог отладки, откуда сможете управлять игрой. Подробное описание модуля смотрите здесь.

14 Создание тем для SDL-INSTEAD

Графический интерпретатор поддерживает механизм тем.

Тема – это инструкция к оформлению игры. Она задаёт внешний вид и положения всех информационных блоков на экране.

Тема представляет из себя каталог, с файлом `theme.ini` внутри. Файл `theme.ini` здесь является ключевым.

Тема, которая является минимально необходимой – это тема `default`. Эта тема всегда загружается первой. Все остальные темы наследуются от неё и могут частично или полностью заменять её параметры. Выбор темы осуществляется пользователем через меню настроек, однако конкретная игра может содержать собственную тему и таким образом влиять на свой внешний вид. В этом случае в каталоге с игрой должен находиться свой файл `theme.ini`. Тем не менее, пользователь свободен отключить данный механизм, при этом интерпретатор будет предупреждать о нарушении творческого замысла автора игры.

Синтаксис `theme.ini` очень прост.

```
<параметр> = <значение>
; комментарий
```

Кроме того, тема может включать в себя другую тему с помощью выражения вида:
`include = имя темы`.
Значения могут быть следующих типов: строка, цвет, число.
Цвет задаётся в форме `#rgb`, где `r`, `g` и `b` – компоненты цвета в шестнадцатеричном виде. Кроме того некоторые основные цвета распознаются по своим именам (см. таблицу 1)

14.1 Параметры окна сцены

Окно сцены – область, в которой располагается сцена. Интерпретация зависит от режима расположения. (см. таблицу 2)
Параметр `scr.gfx.mode` может принимать одно из значений: `fixed`, `embedded` или `float`.
В режиме `embedded` картинка является частью содержимого главного окна, параметры главного окна (см. ниже) `scr.gfx.x`, `scr.gfx.y`, `scr.gfx.w` игнорируются.
В режиме `float` картинка расположена по указанным координатам (`scr.gfx.x`, `scr.gfx.y`) и масштабируе к размеру `scr.gfx.w` x `scr.gfx.h` если превышает его.
В режиме `fixed` – картинка является частью сцены как в режиме `embedded`, но не скроллируется вместе с текстом, а расположена непосредственно над ним.

14.2 Параметры главного окна

Главное окно – область, в которой располагается описание сцены. (табл. 3)
Параметры win.gfx.* оставлены для совместимости. Их не стоит использовать.
Координаты скроллеров могут быть равны -1.

14.3 Параметры области инвентаря

См. таблицу 4.
Координаты скроллеров могут быть равны -1.
Параметр `inv.mode` может принимать значение `horizontal` или `vertical`.
В горизонтальном режиме инвентаря в одной строке могут быть несколько предметов. В вертикальном режиме, в каждой строке инвентаря содержится только один предмет.

14.4 Параметры главного меню

Параметры главного меню INSTEAD-SDL перечислены в таблице 5.

14.5 Прочее

Кроме того, заголовок темы может включать в себя комментарии с тегами. На данный момент существует только один тег: `$Name:`, содержащий строку с именем темы. Например:

```
; $Name:Новая тема$
; модификация темы book
include = book
scr.gfx.h = 500
```

Еще один параметр темы: `snd.click = путь к звуку щелчка`.

Интерпретатор ищет доступные темы в следующих каталогах.

Unix версия интерпретатора просматривает игры в:
`/usr/local/share/instead/themes` (по умолчанию),
`~/.instead/themes`.
WinXP версия:
`Documents and Settings/USER/Local Settings/Application Data/instead/themes`
WinVista: `Users\USER\AppData\Local\instead\themes`
Все Windows: `куда-вы-установили-INSTEAD/themes`

Игра может задавать собственную тему; для этого в каталоге с игрой должен лежать тот самый `theme.ini`. Его формат никак при этом не меняется, просто эта тема загружается сразу после default темы вместе с игрой.

15 Модули

Начиная с версии 1.2.0, INSTEAD также распространяется с некоторым набором полезных модулей. Использование этих модулей очень облегчает разработку игр на INSTEAD.

Модуль подключается при помощи следующей конструкции:

```
require "имя модуля"
```

в начале файла main.lua. При этом важен порядок подключения: некоторые модули могут вызывать ошибки, будучи подключёнными в неправильном порядке.

15.1 Click

Модуль `click` позволяет перехватывать щёлканья мышкой по иллюстрации к сцене.

При клике вызывается обработчик `click` текущей сцены, либо универсальный обработчик `game.click`.

Обработчик получает три параметра: объект (комната или `game`) и координаты клика. Координаты — это абсцисса (`x`) и ордината (`y`) в системе координат оригинального (не масштабированного) изображения. То есть, если оригинальная иллюстрация имеет размер 1024x768 пикселов, но SDL-INSTEAD запущен в полноэкранном режиме 800x600, то координаты будут всё равно вычисляться в системе 1024x768. Иначе код игры пришлось бы переписывать заново для каждого возможного размера картинки.

Координаты считаются от верхнего левого угла.

```
game.click = function(scene, x, y)
 p ("Click at:",x,",", y);
end

house = room {
 nam = 'Дом';
 pic = 'house.png';
 click = function(scene, x, y)
  if x > 100 and x < 120 and y > 50 and y < 90 then
   walk 'street'
  end
 end
}
```

15.2 Dbg

Подключение этого модуля добавляет в инвентарь объект `debug`. При нажатии клавиши F7 или выборе этого объекта вы попадаете в меню, из которого можете:

- переходить между локациями
- оперировать инвентарём (подбирать и бросать предметы)
- выполнять произвольный код Lua
- следить за состоянием объектов

Модуль является сильнейшим средством обмана игры и не рекомендуется для включения на уже выпущенных играх.

15.3 Format, quotes, para, dash

Модуль `format` служит для переформатирования выводимого текста согласно русской книгопечатной традиции.

Замена происходит только при выводе содержимого сцены. Замена не производится в выводе инвентаря и меню.

Модуль имеет всего 4 настройки:

Функция `format.filter` служит для задания пользовательского фильтра. Она получает текст, который будет выведен игроку.

Если настройка `format.para` установлена в `true`, то в начало каждого параграфа будет вставлен небольшой отступ. Например, как в этом справочнике. Вы можете подключить модуль `para` вместо включения этой настройки.

Если настройка `format.dash` установлена в `true`, то каждая последовательность символов `--` будет заменена на среднее тире (–). Вы можете подключить модуль `dash` вместо включения этой настройки.

Наконец, если настройка `format.quotes` установлена в `true`, то будут производиться следующие замены: см. табл. 6

В соответствии с русской типографской традицией, вложенные кавычки должны быть другого стиля: «кавычки „кавычки внутренние” внешние». Тем не менее, при использовании `_"` и `"_` , результат не предсказуем до конца и может быть таким: «внешние кавычки «кавычки внутренние», что также допустимо.

Вы можете подключить модуль `quotes` вместо включения этой настройки.

15.4 Hideinv

Модуль позволяет временно отключать инвентарь. Например, вы хотите вставить в середину игры сцену-заставку, в которой от игрока не требуется управление инвентарём. Для этого эта сцена должна быть определена с атрибутом `hideinv` в значении `true`:

```
require "hideinv"
happyend = room {
    nam = 'Конец';
    hideinv = true;
    dsc = [[ Вы прошли игру! ]];
}
```

Если вы хотите использовать `hideinv` не в обычной комнате, а в `xroom`, то подключите модуль `hideinv` перед модулем `xact`.

15.5 Hotkeys

При подключении этого модуля становится возможным выбирать реплики в диалогах при помощи клавиш 1-9. Так, первая реплика выбирается клавишей 1, вторая – клавишей 2 и т.д. Настроек модуль не имеет.

15.6 Kbd

Модуль `kbd` служит для перехвата событий с клавиатуры.

Он содержит две функции:

Функция `hook_keys` ставит событие перехвата на клавиши, которые переданы ей в качестве параметров. Событие возникает, когда нажата **любая** из указанных клавиш. При возникновении события выполняется блок here().kbd или же, если он отсутствует, блок game.kbd. Выполняющемуся блоку передаются три параметра: self (локация или game), булево значение нажатия (true означает нажато, false – отжато) и текст клавиши.Например:

```
hook_keys('a', 'b', 'c')
```

Событие here().kbd будет вызвано при нажатии любой из клавиш a, b или c.

Клавиши передаются функции соответственно своим клавиатурным кодам, определённым в SDL. Для удобства некоторые коды клавиш приведены в табл. 7

Чтобы снять перехват с клавиш, следует вызвать функцию `unhook_keys`. Её синтаксис аналогичен предыдущей функции, значение же противоположно.

Примером использования модуля `kbd` также служит модуль `hotkeys`.

15.7 Prefs

Модуль `prefs` позволяет сохранять настройки игры. Сохранённые с использованием этого модуля переменные не будут потеряны при начале новой игры.

Загрузка настроек выполняется автоматически при инициализации игры (перед вызовом функции `init()`).

`prefs` – это объект, все переменные которого будут сохранены.

Методы этого объекта:

prefs:store() – сохраняет `prefs`

prefs:purge() – удаляет сохранённый `prefs`

prefs:load() – ручная загрузка `prefs`

15.8 Snapshots

Снапшот — это сохранённое состояние игры.

От обычного сохранения он отличается только тем, что он управляется только разработчиком игры. Вы можете создавать сколько угодно снапшотов (тогда как сохранений может быть только 5). Пользователь может удалять снапшоты (зная, где они лежат и как выглядят), но не может их редактировать.

Сохранения можно вызывать из игры, но нельзя загружать. Сохранения доступны глобально, снапшоты — локально. Пользователь может сделать резервную копию сохранений, но не снапшотов.

Чтобы создать снапшот, следует вызвать функцию `make_snapshot()`. Необязательным параметром является номер слота.

Следует помнить, что снапшот будет создан не моментально, а после завершения текущего такта игры. Иначе сохранённое состояние может оказаться противоречивым.

Чтобы загрузить сохранённый снапшот, нужно вызвать функцию `restore_snapshot()`. Опять же, ей можно передать номер загружаемого слота.

Чтобы удалить сохранённый снапшот, вызывается функция `delete_snapshot()`. Снапшоты занимают место на диске, и их чистка всегда полезна.

```
house = room {
 nam = 'У здания',
 entered = code [[ make_snapshot() ]],
 dsc = 'Вы стоите перед зданием.',
}
```

15.9 Timer

Модуль `timer` – это удобная прослойка для управления таймером.

Метод `timer:set(интервал)` включает таймер с заданным интервалом в миллисекундах. Каждое заданное количество миллисекунд будет вызываться событие `here().timer` либо `game.timer`, если в данной локации не определён этот атрибут. Чтобы отключить таймер, нужно вызвать метод `timer:stop()`.

Главным отличием использования модуля Timer от обычного использования объекта `timer` является то, что при использовании модуля обработчик работает в контексте stead, а не sdl-instead; то есть, вы защищены от ошибок с неправильным возвратом управления.

```
game.timer = function(s)
    set_sound('gfx/beep.ogg');
    p "Timer:"
    p (time())
end
init()
    timer:set(1000)
end
```

15.10 Theme

Модуль `theme` позволяет менять тему в процессе игры, без редактирования файла `main.lua`.

Он объявляет следующие функции:

theme.get('имя параметра') — чтение текущих параметров текущей темы

win.geom(x, y, w, h) — смена координат и размеров главного окна

win.color(fg, link, alink) — смена цветов текста, ссылок и активных ссылок главного окна

win.font(name, size, height) — смена параметров шрифта главного окна

win.gfx.up(pic, x, y) — смена параметров верхнего скроллера главного окна

win.gfx.down(pic, x, y) — смена параметров нижнего скроллера главного окна

inv.geom(x, y, w, h) — смена координат и размеров инвентаря

inv.color(fg, link, alink) — смена цветов текста, ссылок и активных ссылок инвентаря

inv.font(name, size, height) — смена параметров шрифта инвентаря

inv.gfx.up(pic, x, y) — смена параметров верхнего скроллера инвентаря

inv.gfx.down(pic, x, y) — смена параметров нижнего скроллера инвентаря

inv.mode(mode) — смена режима инвентаря

menu.bw(w) — смена толщины границы меню

menu.color(fg, link, alink) — смена цветов текста, ссылок и активных ссылок меню

menu.font(name, size, height) — смена параметров шрифта меню

menu.gfx.button(pic, x, y) — смена параметров значка меню

gfx.cursor(norm, use, x, y) — смена параметров курсора

gfx.mode(mode) — смена режима расположения

gfx.pad(pad) — смена отступов к скролл-барам и краям меню

gfx.bg(bg) — смена фонового изображения

snd.click(name) — смена звукового эффекта для действия

Если часть параметров менять не следует, то их можно заменить значением `nil`.
Пример:

```
gfx.bg "dramatic_bg.png";
win.geom (nil,nil, theme.get 'scr.w', theme.get 'scr.h');
inv.mode 'disabled'
```

15.11 Xact

Довольно много людей жаловались, что в INSTEAD очень неудобно вставлять описание объектов в описание комнат. Например, автор хочет сделать что-то подобное:

> Вы входите в большой зал. На стенах висят картины. С потолка свешивается огромная люстра. Здесь очень красиво, хотя странный сильный запах немного портит впечатление.

Чтобы написать нечто подобное в стандартном окружении INSTEAD, необходимо объявить три объекта: картины, люстра и запах. Описание локации (атрибут dsc) будет пустым, а то, что будет выводиться игроку, будет составляться из описаний предметов. Так, первые две фразы будут храниться в атрибуте dsc объекта «картины», третья фраза будет принадлежать объекту «люстра», а объект «запах» будет описывать всё остальное. При этом объекты должны добавляться в сцену в строго определённом порядке.

Чтобы описание сцены действительно хранилось в описании сцены, был написан модуль `xact`. Он также позволяет делать ссылки на объекты из других объектов и реакций.

Обычная ссылка INSTEAD выглядит так:

```
dsc = "Вы входите в большой зал. На стенах висят {картины}."
```

xact - ссылка выглядит так:

```
dsc = "Вы входите в большой зал. На стенах висят {pictures|картины}."
```

Здесь объект pictures отвечает за картины. Ссылка на него может стоять где угодно. Ссылаться можно на сам объект или его `nam`.

Общий вид ссылок:

`{объект(параметры)|текст}`

До версии 1.2.2 символом разделителя было двоеточие (:). После версии 1.2.2 символ разделителя задаётся полем `stead.delim`; по умолчанию он задан вертикальной линией (|).

Также модуль `xact` содержит несколько упрощённых реализаций объектов:

Объект `xact` описывает простейшую реакцию. Это объект — декорация, он может только выполнять одну функцию, когда к нему обращаются. Объект задаётся так:

```
hello = xact('nam',code[[do_something()]])
```

Первый параметр функции `xact` – имя объекта, второй – реакция. Реакция может быть строкой, функцией или `code`.

Комната `xroom` отличается от обычной тем, что имеет атрибут `xdsc`. Если его задать, то он будет выведен после описания сцены, вместо описаний объектов.

Функцию `xdsc('имя атрибута')` можно вызывать несколько раз, чтобы вывести на экран дополнительные описания.

```
main = room {
 dsc = [[Я в комнате.]];
 xdsc = [[ Я вижу {apple|яблоко} и {knife|нож}. ]];
 other = [[ Еще здесь лежат {chain|цепь} и {tool|пила}.]];
 obj = {
  xdsc(),
  xdsc 'other',
  xact('apple',[[Красное наливное яблочко.]]),
  'knife', 'chain', 'tool',
 }
}
```

16 Дополнительные источники документации

Вот и закончен справочник по INSTEAD. Напомню, что INSTEAD расшифровывается (и переводится) как «Интерпретатор простых текстовых приключений». Официальная документация находится в каталоге doc и поставляется с instead. Дополнительную информацию вы можете получить в Интернете:

- Исходный код
- Сайт программы

Кроме того, полезно будет посмотреть Subversion-репозиторий INSTEAD, где хранится исходный код самого движка и код нескольких полезных трюков, которые не пакуются в релиз для Windows[4]. Вы также можете найти несколько полезных руководств (в том числе и это) в подкаталоге doc/ программы.

[4]Так как релиз для Linux выходит в исходных кодах и заведомо меньше по размеру, то дополнительные исходные коды из него не вырезаются.

Параметр		Параметр		Параметр		Параметр	
aliceblue	T.	forestgreen	T.	mediumvioletred	T.	violet	T.
antiquewhite	T.	fuchsia	T.	midnightblue	T.	violetred	T.
aqua	T.	gainsboro	T.	mintcream	T.	wheat	T.
aquamarine	T.	ghostwhite	T.	mistyrose	T.	white	T.
azure	T.	gold	T.	moccasin	T.	whitesmoke	T.
beige	T.	goldenrod	T.	navajowhite	T.	yellow	T.
bisque	T.	gray	T.	navy	T.	yellowgreen	T.
black		green	T.	oldlace	T.		
blanchedalmond	T.	greenyellow	T.	olive	T.		
blue	T.	honeydew	T.	olivedrab	T.		
blueviolet	T.	hotpink	T.	orange	T.		
brown	T.	indianred	T.	orangered	T.		
burlywood	T.	indigo	T.	orchid	T.		
cadetblue	T.	ivory	T.	palegoldenrod	T.		
chartreuse	T.	lavender	T.	palegreen	T.		
chocolate	T.	lavenderblush	T.	paleturquoise	T.		
coral	T.	lawngreen	T.	palevioletred	T.		
cornflowerblue	T.	lemonchiffon	T.	papayawhip	T.		
cornsilk	T.	lightblue	T.	peachpuff	T.		
crimson	T.	lightcoral	T.	peru	T.		
cyan	T.	lightcyan	T.	pink	T.		
darkblue	T.	lightgoldenrodyellow	T.	plum	T.		
darkcyan	T.	lightgrey	T.	powderblue	T.		
darkgoldenrod	T.	lightgreen	T.	purple	T.		
darkgray	T.	lightpink	T.	red	T.		
darkgreen	T.	lightsalmon	T.	rosybrown	T.		
darkkhaki	T.	lightseagreen	T.	royalblue	T.		
darkmagenta	T.	lightskyblue	T.	saddlebrown	T.		
darkolivegreen	T.	lightslateblue	T.	salmon	T.		
darkorange	T.	lightslategray	T.	sandybrown	T.		
darkorchid	T.	lightsteelblue	T.	seagreen	T.		
darkred	T.	lightyellow	T.	seashell	T.		
darksalmon	T.	lime	T.	sienna	T.		
darkseagreen	T.	limegreen	T.	silver	T.		
darkslateblue	T.	linen	T.	skyblue	T.		
darkslategray	T.	magenta	T.	slateblue	T.		
darkturquoise	T.	maroon	T.	slategray	T.		
darkviolet	T.	mediumaquamarine	T.	snow	T.		
deeppink	T.	mediumblue	T.	springgreen	T.		
deepskyblue	T.	mediumorchid	T.	steelblue	T.		
dimgray	T.	mediumpurple	T.	tan	T.		
dodgerblue	T.	mediumseagreen	T.	teal	T.		
feldspar	T.	mediumslateblue	T.	thistle	T.		
firebrick	T.	mediumspringgreen	T.	tomato	T.		
floralwhite	T.	mediumturquoise	T.	turquoise	T.		

Table 1: Цветовые константы INSTEAD

Параметр	Тип	Описание
scr.w	число	ширина игрового пространства, пиксели
scr.h	число	высота игрового пространства, пиксели
scr.col.bg	цвет	цвет фона
scr.gfx.bg	строка	путь к файлу фонового изображения
scr.gfx.cursor.x	число	абсцисса центра курсора, пиксели
scr.gfx.cursor.y	число	ордината центра курсора, пиксели
scr.gfx.cursor.normal	строка	путь к картинке-курсору
scr.gfx.cursor.use	строка	путь к картинке-курсору режима использования
scr.gfx.use	строка	путь к картинке-индикатору режима использования
scr.gfx.pad	число	размер отступов к скролл-барам и краям меню, пиксели
scr.gfx.x	число	абсцисса окна изображений, пиксели
scr.gfx.y	число	ордината окна изображений, пиксели
scr.gfx.w	число	ширина окна изображений, пиксели
scr.gfx.h	число	высота окна изображений, пиксели
scr.gfx.mode	строка	режим расположения

Table 2: Параметры окна изображений

Параметр	Тип	Описание
win.x	число	абсцисса главного окна, пиксели
win.y	число	ордината главного окна, пиксели
win.w	число	ширина главного окна, пиксели
win.h	число	высота главного окна, пиксели
win.fnt.name	строка	путь к файлу шрифта
win.fnt.height	число	междустрочный интервал
win.fnt.size	число	размер шрифта главного окна, пункты
win.gfx.up	строка	путь к файлу изображения скроллера вверх для главного окна
win.gfx.down	строка	путь к файлу изображения скроллера вниз для главного окна
win.gfx.w	число	синоним scr.gfx.w
win.gfx.h	число	синоним scr.gfx.h
win.col.fg	цвет	цвет текста главного окна
win.col.link	цвет	цвет ссылок главного окна
win.col.alink	цвет	цвет активных ссылок главного окна
win.up.x	число	абсцисса верхнего скроллера
win.up.y	число	ордината верхнего скроллера
win.down.x	число	абсцисса нижнего скроллера
win.down.y	число	ордината нижнего скроллера

Table 3: Параметры главного окна

Параметр	Тип	Описание
inv.x	число	абсцисса области инвентаря,пиксели
inv.y	число	ордината области инвентаря,пиксели
inv.w	число	ширина области инвентаря,пиксели
inv.h	число	высота области инвентаря,пиксели
inv.col.fg	цвет	цвет текста инвентаря
inv.col.link	цвет	цвет ссылок инвентаря
inv.col.alink	цвет	цвет активных ссылок инвентаря
inv.fnt.name	строка	путь к шрифту инвентаря
inv.fnt.height	число	междустрочный интервал
inv.fnt.size	число	размер шрифта инвентаря,пункты
inv.gfx.up	строка	путь к изображению скроллера вверх для инвентаря
inv.gfx.down	строка	путь к изображению скроллера вниз для инвентаря
inv.mode	строка	режим инвентаря
inv.up.x	число	абсцисса верхнего скроллера
inv.up.y	число	ордината верхнего скроллера
inv.down.x	число	абсцисса нижнего скроллера
inv.down.y	число	ордината нижнего скроллера

Table 4: Параметры области инвентаря

Параметр	Тип	Описание
menu.col.bg	цвет	цвет фона меню
menu.col.fg	цвет	цвет текста меню
menu.col.link	цвет	цвет ссылок меню
menu.col.alink	цвет	цвет активных ссылок меню
menu.col.alpha	цвет	прозрачность меню (0–255)
menu.col.border	цвет	цвет границы меню
menu.bw	число	толщина границы меню, пиксели
menu.fnt.name	строка	путь к шрифту меню
menu.fnt.size	число	размер шрифта меню, пункты
menu.gfx.button	строка	путь к значку меню
menu.button.x	число	абсцисса кнопки меню, пиксели
menu.button.y	число	ордината кнопки меню, пиксели
menu.fnt.height	число	междустрочный интервал

Table 5: Параметры главного меню

Последовательность символов	Результат
<<	«
>>	»
,,	„
'' (два апострофа)	”
_"	«
"_	»
"	« или », угадывается автоматически

Table 6: Замены кавычек

Ключ	Клавиша	Ключ	Клавиша
a	Английская «a»	[/]	/ на цифровой клавиатуре
b	Английская «b»	[*]	* на цифровой клавиатуре
c	Английская «c»	[-]	- на цифровой клавиатуре
...	...	[+]	+ на цифровой клавиатуре
z	Английская «z»	enter	Enter на цифровой клавиатуре
0	0	[0]	0 на цифровой клавиатуре
1	1	[1]	1 на цифровой клавиатуре
2	2	[2]	2 на цифровой клавиатуре
3	3	[3]	3 на цифровой клавиатуре
...	...	...	...
9	9	[9]	9 на цифровой клавиатуре
return	Enter	[.]	. на цифровой клавиатуре
escape	Esc	left ctrl	Левый Ctrl
backspace	Backspace	left shift	Левый Shift
tab	Tab	left alt	Левый Alt
space	Пробел	right ctrl	Правый Ctrl
-	-	right shift	Правый Shift
=	=	right alt	Правый Alt
[	Английская «[»	numlock	Num Lock
]	Английская «]»	caps lock	Caps Lock
\	\	scroll lock	Scroll Lock
;	Английская «;»		
’	Английская «’»		
‘	Английская «‘»		
,	Английская «,»		
.	Английская «.»		
/	Английская «/»		
f1	F1		
f2	F2		
f3	F3		
...	...		
f12	F12		
print screen	Print Screen		
pause	Pause		
insert	Insert		
home	Home		
page up	Page Up		
delete	Delete		
end	End		
page down	Page Down		
right	Right (правая стрелка курсора)		
left	Left (левая стрелка курсора)		
down	Down (стрелка курсора вниз)		
up	Up (стрелка курсора вверх)		

Table 7: Ключи некоторых часто употребляемых клавиш в SDL

Index