Table of Contents

Инструкция

API

Lua / MoonScript

JavaScript

Прочее

TIC-80

TIC-80 это виртуальная игровая консоль, для которой Вы можете создавать игры, играть в них и делиться ими. Имеются встроенные средства разработки: **редакторы кода**, **спрайтов**, **карт** , **звуковых эффектов** и **музыки**, а также **командная строка**, которых достаточно для создания **ретро мини игры**. На выходе Вы получите файл картриджа, который можно сохранить и запустить на сайте. Также игра может быть упакована в плеер, который запускается на всех популярных платформах, и распространена по Вашему желанию. Весь процесс создания игры в стиле ретро проходит с некоторыми техническими ограничениями: дисплей **240x136** пикселей, палитра из **16 цветов**, **256 цветных спрайтов 8x8**, **4-х канальный звук** и прочее.

- status: в разработке version 0.47.0
- website: https://tic.computer
- twitter: @tic_computer
- hashtag: #TIC80
- chats: Gitter / Discord
- community: Itch.io / VKontakte
- bugs: Github
- wiki: Github
- changelog: Github
- author: Vadim Grigoruk // grigoruk@gmail.com

Спецификация

Экран: 240x136 пикселей, палитра 16 цветов
Ввод: 2 джойстика с восемью кнопками (крестовина + 4 кнопки) либо мышь/сенсорный ввод
Спрайты: 256 спрайтов 8x8 переднего плана и 256 тайлов 8x8 фона
Карта тайлов: 240x136 ячеек, 1920x1088 пикселей
Звук: 4 канала (с редактируемыми огибающими формы волны)
Код: 64Kb кода на Lua / Moonscript / JavaScript

Схема памяти 80Kb

```
+-----------------------------------+
|          80K RAM LAYOUT           |
+-------+-------------------+-------+
| ADDR  | INFO              | SIZE  |
+-------+-------------------+-------+
| 00000 | SCREEN            | 16320 | 240x136 = 32640 4bit pixels
| 03FC0 | PALETTE           | 48    | 16 x 24bit RGB color values
V 03FF0 | PALETTE MAP       | 8     | 16 x 4bit color indexes
R 03FF8 | BORDER COLOR      | 1     | 4bit color value
A 03FF9 | SCREEN OFFSET     | 2     | horz/vert screen offset [-127...+127]
M 03FFB | GAMEPAD MASK      | 1     |
| 03FFC | GAMEPAD STATE     | 2     | x 2 gamepads, 1bit for UP DOWN LEFT RIGHT A B X Y
| 03FFE | ...               | 2     |
+-------+-------------------+-------+
| 04000 | SPRITES           | 16384 | 256 8x8 4bit sprites
| 06000 | TILES             | 16384 | 256 8x8 4bit bg tiles
| 08000 | MAP               | 32640 | 240x136 map cells
| 0FF80 | PERSISTENT MEMORY | 28    |
| 0FF9C | SOUND REGISTERS   | 72    | ...
| 0FFE4 | WAVEFORMS         | 256   | 16 waveforms, each 32 x 4bit values
| 100E4 | SFX               | 4224  | ...
| 11164 | MUSIC PATTERNS    | 11520 | ...
| 13E64 | MUSIC TRACKS      | 408   | ...
| 13FFC | MUSIC POS         | 4     | ...
| 14000 | ...               | 0     |
+-------+-------------------+-------+


+-----------------------+
|    SOUND REGISTER     |
+-----------+-----------+
| frequency | 12bit     | x 4 channels = 72 bytes
| volume    | 4bit      |
| waveform  | 32 x 4bit |
+-----------+-----------+


+---------------------------+
|           SFX             |
+---------------------------+
|                           |
| +----------+------+       |
| | volume   | 4bit |       |
| | wave     | 4bit | x 30  |
| | arpeggio | 4bit |       |
| | pitch    | 4bit |       |
| +----------+------+       |
|                           |
| +----------+------+       |
| | octave   | 3bit |       | x 64 = 4224 bytes
| | pitch16x | 1bit |       |
| | speed    | 3bit |       |
| | reverse  | 1bit |       |
| | note     | 4bit |       |
| | temp     | 4bit |       |
| +----------+------+       |
|                           |
| +--------------+          |
| |     LOOP     |          |
| +-------+------+ x 4      |
| | start | 4bit |          |
```

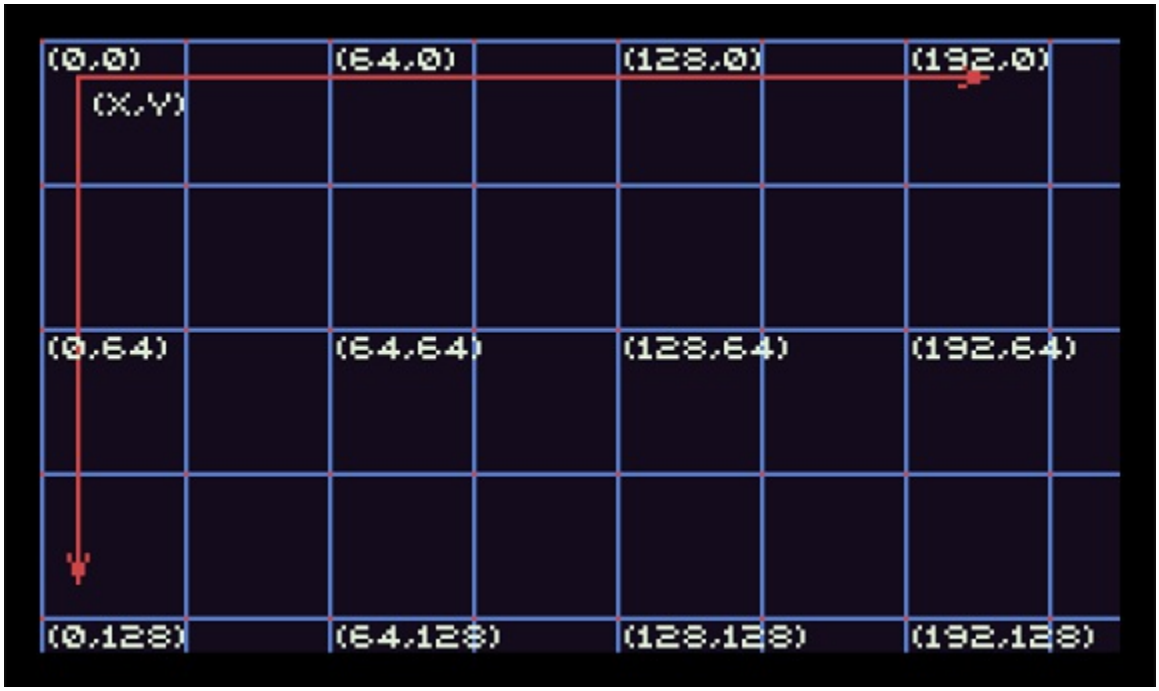
```
| | size  | 4bit |         |
| +-------+------+         |
|                          |
+--------------------------+


+------------------------+
|      MUSIC PATTERN     |
+------------------------+
|                        |
| +---------------+      |
| |  PATTERN ROW  |      |
| +--------+------+      |
| | note   | 4bit |      |
| | volume | 4bit |      |
| | param  | 4bit | x 64 | x 60 = 11520 bytes
| | effect | 3bit |      |
| | sfx    | 6bit |      |
| | octave | 3bit |      |
| +--------+------+      |
|                        |
+------------------------+


+----------------------------+
|        MUSIC TRACK         |
+----------------------------+
|                            |
| +------------+------+      |
| | pattern #0 | 6bit |      |
| | pattern #1 | 6bit | x 16 |
| | pattern #2 | 6bit |      |
| | pattern #3 | 6bit |      |
| +------------+------+      | x 8 = 408 bytes
|                            |
| +-------+------+           |
| | tempo | 8bit |           |
| | rows  | 8bit |           |
| | speed | 8bit |           |
| +-------+------+           |
|                            |
+----------------------------+

+-----------------+
|     MUSIC POS   |
+-------+---------+
| track | 8bit    |
| frame | 8bit    |
| row   | 8bit    |
+-------+---------+
|                 | = 4 bytes
| +-------------+ |
| |    FLAGS    | |
| +------+------+ |
| | loop | 1bit | |
| | ...  | 7bit | |
| +------+------+ |
|                 |
+-----------------+

empty waveform generates noise by default
you can update Sound Registers every tick (60 times per second)

to read frequency and volume use the following code:
...
value = peek(0xFF9C+18*channel+1)<<8|peek(0xFF9C+18*channel)
```

```
frequency = (value&0x0fff)
volume = (value&0xf000)>>12
...
```

Система координат

Стандартное разрешение экрана TIC составляет 240 пикселей в ширину и 136 пикселей в высоту. Пропорции сохраняются при изменении размера, поэтому из кода работа всегда ведется с «виртуальным» родным разрешением, даже если Вы работаете с монитором 4K в полно-экранном режиме.
Координаты, как и в большинстве компьютерных систем координат, начинаются с верхнего-левого угла экрана.
Координата X - координата по горизонтали, начинается с 0 (левая сторона) до 240 (правая сторона).
Координата Y - координата по вертикали, начинается с 0 (верх) до 136 (низ).

Пример:



Запустить или скачать картридж примера.

```
-- title:  Coordinate system demo
-- author: Filippo
-- desc:   shows coordinate system
-- script: lua
-- input:  gamepad

function TIC()
  cls(0)

  --grid x
  for x=0,240,32 do
    line(x,0,x,136,8)
  end

  --grid y
  for y=0,136,32 do
    line(0,y,240,y,8)
  end

  --grid cross
```

```
  for x=0,240,32 do
    for y=0,136,32 do
      pix(x,y,6)
    end
  end

  --arrows
  line(8,8,210,8,6)
  line(210,8,210-6,8-2,6)
  line(210,8,210-6,8+2,6)

  line(8,8,8,120,6)
  line(8,120,8-2,120-6,6)
  line(8,120,8+2,120-6,6)

  --labels
  for x=0,240,64 do
    for y=0,136,64 do
      print('('..x..','..y..')',x+2,y+2)
    end
  end

  --reference
  print('(X,Y)',12,12)
end
```

Запустить или скачать картридж примера.

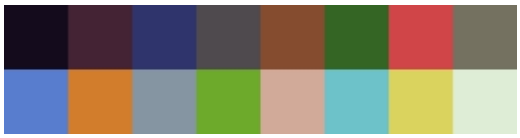
```
-- title:  Coordinate system demo
-- author: Filippo
-- desc:   shows coordinate system
-- script: moon
-- input:  gamepad

export TIC=->
  cls 0

  --grid x
  for x=0,240,32
    line x,0,x,136,8

  --grid y
  for y=0,136,32
    line 0,y,240,y,8

  --grid cross
  for x=0,240,32
    for y=0,136,32
      pix x,y,6

  --arrows
  line 8,8,210,8,6
  line 210,8,210-6,8-2,6
  line 210,8,210-6,8+2,6

  line 8,8,8,120,6
  line 8,120,8-2,120-6,6
  line 8,120,8+2,120-6,6

  --labels
  for x=0,240,64
    for y=0,136,64
      print '('..x..','..y..')',x+2,y+2
```

```
  --reference
  print '(X,Y)',12,12
```

```
// title:  Coordinate system demo
// author: Filippo
// desc:   shows coordinate system
// script: js
// input:  gamepad

function TIC() {
  cls(0);

  //grid x
  for (var x = 0; x < 240; x += 32) {
    line(x, 0, x, 136, 8);
  }

  //grid y
  for (var y = 0; y < 136; y += 32) {
    line(0, y, 240, y, 8);
  }

  //grid cross
  for (var x = 0; x < 240; x += 32) {
    for (var y = 0; y < 136; y += 32) {
      pix(x, y, 6);
    }
  }

  //arrows
  line(8, 8, 210, 8, 6);
  line(210, 8, 210 - 6, 8 - 2, 6);
  line(210, 8, 210 - 6, 8 + 2, 6);

  line(8, 8, 8, 120, 6);
  line(8, 120, 8 - 2, 120 - 6, 6);
  line(8, 120, 8 + 2, 120 - 6, 6);

  //labels
  for (var x = 0; x < 240; x += 64) {
    for (var y = 0; y < 136; y += 64) {
      print('(' + x + ',' + y + ')', x + 2, y + 2);
    }
  }

  //reference
  print('(X,Y)', 12, 12);
}
```

Цветовая палитра

Стандартной цветовой палитрой TIC-80 является `DB16`:



Имя цвета	Индекс	Шестнадцатеричное значение
Чёрный	0	140C1C
Тёмно-красный	1	442434
Тёмно-синий	2	30346D
Тёмно-серый	3	4E4A4F
Коричневый	4	854C30
Тёмно-зеленый	5	346524
Красный	6	D04648
Светло-серый	7	757161
Светло-синий	8	597DCE
Оранжевый	9	D27D2C
Сине-серый	10	8595A1
Светло-зелёный	11	6DAA2C
Персиковый	12	D2AA99
Голубой	13	6DC2CA
Жёлтый	14	DAD45E
Белый	15	DEEED6

Вы можете настроить палитру с помощью команды `poke`

Вы можете устанавливать палитру настраивая цвета в **редакторе спрайтов** перемещая ползунки, либо вставив строку, содержащую цвета в шестнадцатеричном виде.
Например такую: `000000574200403l8d5050508b542955a0498839327878788b3f967869c49f9f9f94e089b8696267b6bdbfce72ffffff`

Пример:

Запустить или скачать картридж примера.

```
-- title:  palette
-- author: Nesbox
-- desc:   how to switch palatte in runtime demo
-- script: lua
-- input:  gamepad

local W=240
local H=136

-- palette address
local ADDR=0x3FC0*2

local PALETTES = {
  {name="DB16",   data="deeed6dad45e6dc2cad2aa996daa2c8595a1d27d2c597dce757161d046483465246 54c304e4a4e30346d442434140
c1c"},
  {name="PICO-8", data="fff1e8fff02429adffffccaa00e756c2c3c7ffa300ff77a883769cff004d008751ab52365f574f1d2b537e2553000
000"},
  {name="ARNE16", data="fffffff7e26bb2dcefe06f8ba3ce279d9d9deb893131a2f22f484ebe263344891aa46422493c2b0057841b2632000
000"},
  {name="EDG16",  data="fffffffe7622ce8f4e4a67263c64dafbfd2fb922b0484d14f6781e53b44327345b86f509e2835743f393f2832193
d3f"},
  {name="A64",    data="ede6c8bbc8408fbfd5cd93739ccc479cabb18385cf7655a2808078b1486350945092562b485454313a914c3435000
000"},
  {name="C64",    data="fffffbfce7267b6bdb8696294e0899f9f9f7869c48b3f9678787888393255a0498b542950505040318d574200000
000"},
}

local t=0
local index=1

-- update palette
function updpal()
  local pal=PALETTES[index].data
  for i=1,#pal do
    poke4(ADDR+#pal-i,tonumber(pal:sub(i,i),16))
  end
end

updpal()

function TIC()
```

```
  -- handle input
  if btnp(0,30,6) and index>1 then
    index=index-1
    updpal()
  end
  if btnp(1,30,6) and index<#PALETTES then
    index=index+1
    updpal()
  end

  if btnp(4)or btnp(5)then exit()end

  -- draw

  cls(15)

  print("SELECT PALETTE",6,6,0)

  for i,v in pairs(PALETTES) do
    print(v.name,12,12+i*6,0)
  end

  print(">",6+(t//15%2),12+index*6,0)

  local S=16

  -- draw palette
  for i=0,15-1 do
    for j=0,S-1 do
      line(W-j-i*S,H,W,H-j-i*S,i)
    end
  end

  t=t+1
end
```

Запустить или скачать картридж примера.

```
-- title:  palette
-- author: Nesbox
-- desc:   how to switch palatte in runtime demo
-- script: moon
-- input:  gamepad

W=240
H=136

-- palette address
ADDR=0x3FC0*2

PALETTES = {
  {name:"DB16",   data:"deeed6dad45e6dc2cad2aa996daa2c8595a1d27d2c597dce757161d046483465248 54c304e4a4e30346d442434140c1c"},
  {name:"PICO-8", data:"fff1e8fff02429adffffccaa00e756c2c3c7ffa300ff77a883769cff004d008751ab52365f574f1d2b537e2553000000"},
  {name:"ARNE16", data:"ffffffff7e26bb2dcefe06f8ba3ce279d9d9deb893131a2f22f484ebe263344891aa46422493c2b0057841b2632000000"},
  {name:"EDG16",  data:"fffffffe7622ce8f4e4a67263c64dafbfd2fb922b0484d14f6781e53b44327345b86f509e2835743f393f2832193d3f"},
  {name:"A64",    data:"ede6c8bbc8408fbfd5cd93739ccc479cabb18385cf7655a2808078b1486350945092562b485454313a914c3435000000"},
  {name:"C64",    data:"ffffffbfce7267b6bdb8696294e0899f9f9f7869c48b3f9678787888393255a0498b542950505040318d574200000000"},
```

```
}

t=0
index=1

-- update palette
updpal=->
  pal=PALETTES[index].data
  for i=1,#pal
    poke4(ADDR+#pal-i,tonumber(pal\sub(i,i),16))

updpal()

export TIC=->

  -- handle input
  if btnp(0,30,6) and index>1
    index-=1
    updpal()
  if btnp(1,30,6) and index<#PALETTES
    index+=1
    updpal()

  if btnp(4) or btnp(5) then exit()

  -- draw

  cls 15

  print "SELECT PALETTE",6,6,0

  for i,v in pairs(PALETTES)
    print v.name,12,12+i*6,0

  print ">",6+(t//15%2),12+index*6,0

  S=16

  -- draw palette
  for i=0,15-1
    for j=0,S-1
      line W-j-i*S,H,W,H-j-i*S,i

  t=t+1
```

Примеры цветовых палитр для использования в играх.

Все представленные палитры имеют 16 цветов.

```
DB16
deeed6dad45e6dc2cad2aa996daa2c8595a1d27d2c597dce757161d046483465248 54c304e4a4e30346d442434140c1c
```

```
PICO-8
0000001d2b537e2553008751ab52365f574fc2c3c7fff1e8ff004dffa300ffec2700e43629adff83769cff77a8ffccaa
```

```
ARNE16
fffffff7e26bb2dcefe06f8ba3ce279d9d9deb893131a2f22f484ebe263344891aa46422493c2b0057841b2632000000
```

```
EDG16
fffffffe7622ce8f4e4a67263c64dafbfd2fb922b0484d14f6781e53b44327345b86f509e2835743f393f2832193d3f
```

```
A64
ede6c8bbc8408fbfd5cd93739ccc479cabb18385cf7655a2808078b1486350945092562b485454313a914c3435000000
```

```
C64
fffffbfce7267b6bdb8696294e0899f9f9f7869c48b3f9678787888393255a0498b542950505040318d574200000000
```

```
COMMODORE VIC-20 PALETTE
000000ffffffa8734ae9b287772d26b6686285d4dcc5ffffa85fb4e99df5559e4a92df8742348b7e70cabdcc71ffffb0
```

```
STILL LIFE PALETTE
3f28117a2222d13b27e07f8a5d853a68c127b3e8681226155131552 86fb89b8bffa8e4d4cc8218c7b581000000ffffff
```

```
JAPANESE MACHINE PALETTE
00000019102846af45a1d685453e787664fe8331299ec2e8dc534be18d79d6b97be9d8a1216c4bd365c8afaab9f5f4eb
```

```
CGARNE PALETTE
0000005e606e2234d10c7e455c2e78b5b5b5FFFFFFFFd93f7be2f98a36224c81fb44aacceb8a60aa5c3d6cd947e23d69
```

```
PSYGNOSIA PALETTE
0000001b1e29362747443f4152524c64647c73615077785b9ea4a7cbe8f7e08b79a2324e003308084a3c546a00516cbf
```

```
COLOR GRAPHICS ADAPTER PALETTE
000000555555AAAAAAFFFFFF0000AA5555FF00AA0055FF5500AAAA55FFFFAA0000FF5555AA00AAFF55FFAA5500FFFF55
```

```
EROGE COPPER PALETTE
0d080d4f2b24825b31c59154f0bd77fbdf9bfff9e4bebbb27bb24e74adbb4180a032535f2a23497d3840c16c5be89973
```

```
EASTER ISLAND PALETTE
f6f6bfe6d1d1868691794765f5e17aedc38dcc8d86ca657e39d4b98dbcd28184ab6860869dc0857ea788567864051625
```

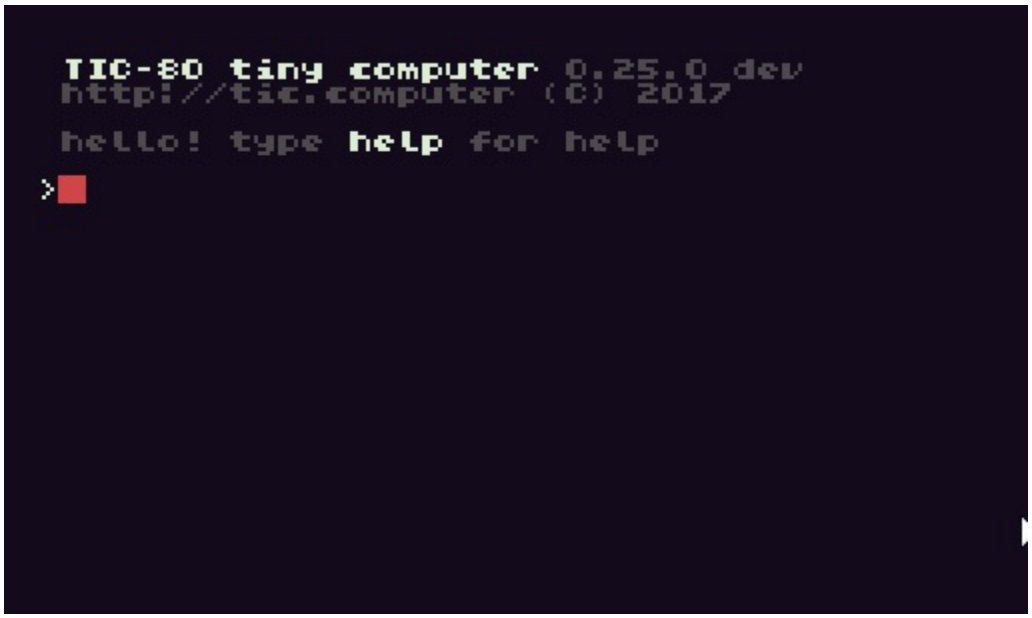
```
ZX-Spectrum Orthodox palette
0000000000cda70000a700cd00b70000b7cda7b700a7b7cd0000000000ffd00000d000ff00e40000e4ffd0e400d0e4ff
```

```
ZX-Spectrum Pulsar palette
0000000000cdcd0000cd00cd00cd0000cdcdcdcd00cdcdcd0000000000ffff0000ff00ff00ff0000ffffffff00ffffff
```

```
ZX-Spectrum Alone palette
0000000000a0a00000a000a000a00000a0a0a0a000a0a0a00000000000ffff0000ff00ff00ff0000ffffffff00ffffff
```

```
ZX-Spectrum Electroscale palette
3e414c4e515f5e62736e73867e839a8e94ad9ea4c1aeb5d43e414c525564666a7c7a7f948e93ada2a8c5b5bcddc9d1f5
```

Консоль



Доступные команды

- `help` - показать доступные команды
- `surf` - команда для просмотра локальных и картриджей с веб-сайта
- `new [lua | moon | js]` - создать новый картридж
- `config default lua` или `config default` - загружает заготовку картриджа создаваемого командой `new` или `new lua`
- `config default moon` или `config default moonscript` - загружает заготовку картриджа создаваемого командой `new moon`
- `config default js` - загружает заготовку картриджа создаваемого командой `new js`
- `load <cart>` - загрузить картридж из рабочей директории (можно писать имя файла без расширения

.tic)

- `load <cart> [sprites | map | cover | code | sfx | music | palette]` - загрузить указанные данные из картриджа
- `save <cart>` - сохранить картридж в рабочую директорию локальной файловой системы
- `run` - запустить текущий проект
- `ls | dir` - показать список файлов в рабочей директории
- `mkdir` - создать директорию
- `cd` - сменить директорию
- `folder` - открыть рабочую директорию в операционной системе (Windows, Linux, MacOSX)
- `add` - показать файловый диалог для добавления файла в рабочую директорию
- `del <file>` - удалить файл из рабочей директории
- `get <file>` - показать файловый диалог для скачивания файла
- `export [html | native | sprites | cover | map]` - экспорт **html**, **нативной сборки** игры, **спрайтов**, **обложки** или **карты тайлов** игры
- `import [sprites | cover | map]` - импорт **спрайтов**, **обложки** из .gif или **карты тайлов**
- `cls` - очистить экран
- `demo` - установить демо картриджи в рабочую директорию
- `config` - загрузить конфигурационный картридж **config.tic**
- `config save` - записать текущую конфигурацию
- `config reset` - сбросить конфигурацию к первоначальным настройкам
- `keymap` - отобразить меню настройки кнопок джойстиков
- `resume` - команда для возврата в игру в то место, на котором она была остановлена
- `ram` - показать схему памяти 80Kb
- `exit` - выйти из приложения

config

конфигурация значений по умолчанию

Пример:

Описание:
Запустив в консоли **TIC-80** команду `config` откроется конфигурационный картридж.
В нём Вы можете настроить системную графику в редакторе спрайтов: системный шрифт, цветовую палитру, эмотиконс, бэк для режима `surf`, кнопки для музыкального редактора, отображение кнопок джойстика для сенсорного ввода.
В редакторе звуковых эффектов: звук при запуске. В редакторе кода:

- индекс спрайта для отображения курсора(-1 не отображать спрайт)
- индексы цветов для отображения кода
- проверка новой версии
- длительность записи gif
- масштаб записи gif

Сохранять изменения можно нажатием комбинации клавиш `CTRL+S`, либо набрав в консоли `config save`
Сбросить конфигурацию к первоначальным настройкам можно при помощи команды `config reset`

Вот стандартный настройки конфигурации:

```
THEME=
{
    CURSOR=
    {
        SPRITE=-1,
        PIXEL_PERFECT=true,
    },

    CODE =
    {
        BG     =1,
```

```
        STRING =14,
        NUMBER =13,
        KEYWORD=6,
        API    =11,
        COMMENT=7,
        SIGN   =10,
        VAR    =15,
        OTHER  =0,
        SELECT =3,
        CURSOR =6,
    },

    GAMEPAD=
    {
        TOUCH=
        {
            ALPHA=180,
        },
    },
}

CHECK_NEW_VERSION=false
GIF_LENGTH=20 -- in seconds
GIF_SCALE=2
```

Встроенные редакторы

В TIC-80 встроены редакторы, позволяющие написать игру с нуля.
Вы можете писать код, рисовать графику, создавать карту уровней, звуковые эффекты и музыку.
Стоит отметить, что все редакторы обладают необходимым базовым функционалом, но не призваны заменить собой сторонние профессиональные инструменты.

Редактор кода

Редактор кода представляет из себя текстовый редактор, имеющий подсветку синтаксиса Lua/MoonScript/JavaScript, а также инструменты для навигации и редактирования кода.

Описание графического интерфейса

Блок кнопок в верху экрана, общий для всех встроенных редакторов.
В левом-верхнем углу экрана расположены кнопки перехода во встроенные редакторы: редактор кода, редактор спрайтов, редактор карты тайлов, редактор звуковых эффектов и редактор музыки. Как и в любом другом встроенном редакторе **TIC-80**, Вы можете выполнить операции: вырезать, скопировать, вставить и отменить/вернуть изменения.
Эти действия могут быть выполнены как по нажатию на соответствующую кнопку в верхней части экрана, так и по 'горячим клавишам'.
Вверху по центру отображается текстовая подсказка для всех кнопок верхнего блока.

Интерфейс редактора кода.
Кнопки в верхней-правой части экрана позволяют выполнить действия: поиск текста, переход к строке и отображение имён функций. Можно также перейти к функции, нажав на её имя в списке имён функций.
В левой-нижней части экрана выводится информация о текущем положении курсора: текущий номер линии из всех доступных и номер колонки.
В правой-нижней части экрана выводится информация о занятых байтах из всей доступной памяти для размещения кода.

Метаданные картриджа

```
-- title:  название игры
-- author: разработчик игры
-- desc:   краткое описание
-- script: lua / moon / js
-- input:  gamepad / mouse
-- saveid: уникальное имя используемое `pmem`
```

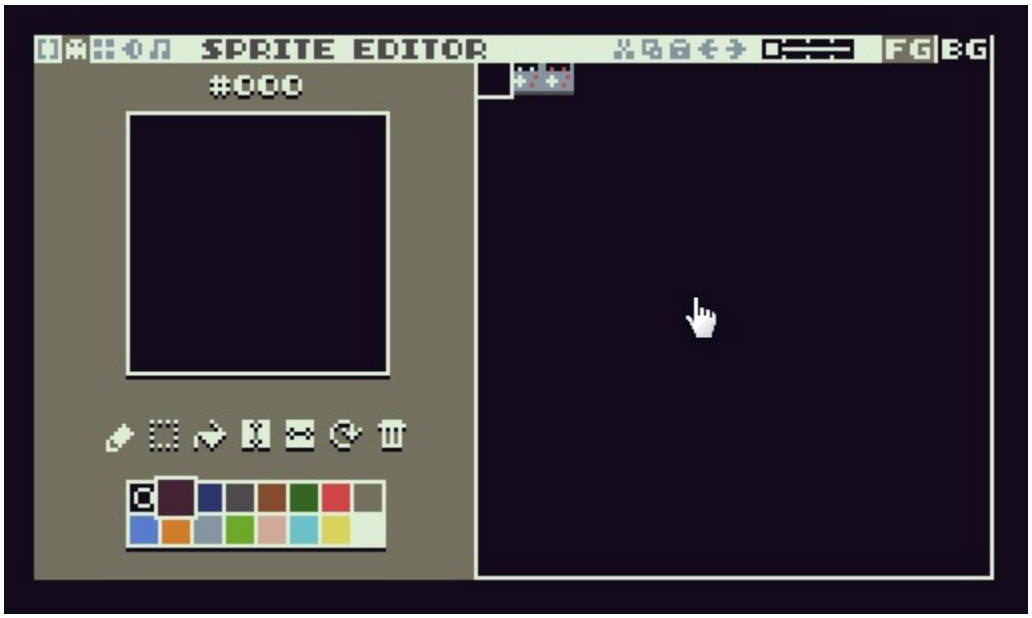
Хорошим тоном является объявление всех этих тэгов в начале кода.
Для поддержки языка **Moonscript** (синтаксис Lua будет недоступен) укажите тэг `-- script: moon`.
Для использования языка **JavaScript** укажите `// scripr: js`
Поддержка **мыши** (отключение джойстиков) `-- input: mouse`
Указанный уникальный savevid для вашей игры будет использовать pmem вместо того, чтобы полагаться на хеш MD5.

Редактор спрайтов

Редактор спрайтов представляет из себя простейший пиксельный редактор, в котором Вы можете рисовать спрайты переднего плана(FG) и спрайты(тайлы) заднего плана(BG).

Определения терминов.
Спрайт представляет собой блок графики 8x8 пикселей.
Тайлом будем называть спрайты размещенные на спрайт-листе графики заднего плана.
Спрайт-лист графики заднего плана(BG): 256 спрайтов с индексами 0-255.
Спрайт-лист графики переднего плана(FG): 256 спрайтов с индексами 256-511.
Любые спрайты из 512 могут быть выведены на экран с помощью функции API `spr`, и только спрайты с индексами 0-255(графика заднего плана) могут быть выведены функцией `map` и расставлены в редакторе карты тайлов.

Описание графического интерфейса



Блок кнопок в верху экрана, общий для всех встроенных редакторов.
В левом-верхнем углу экрана расположены кнопки перехода во встроенные редакторы: редактор кода, редактор спрайтов, редактор карты тайлов, редактор звуковых эффектов и редактор музыки. Как и в любом другом встроенном редакторе **TIC-80**, Вы можете выполнить операции: вырезать, скопировать, вставить и отменить/вернуть изменения.
Эти действия могут быть выполнены как по нажатию на соответствующую кнопку в верхней части экрана, так и по 'горячим клавишам'.
Вверху по центру отображается текстовая подсказка для всех кнопок верхнего блока.

Интерфейс редактора спрайтов.
В правом-верхнем углу экрана располагаются кнопки смены размера области отображения графики: 8x8, 16x16, 32x32 и 64x64 пикселя. Кнопки FG и BG позволяют переключиться на графику переднего и заднего плана.

Остальная область экрана разделена на две части: справа отображается спрайт-лист графики переднего/заднего плана. С левой стороны располагаются инструменты и рабочая область графического редактора.

Спрайт-лист графики переднего/заднего плана.
Рамка на спрайт-листе переднего/заднего плана выделяет ту область, что отображается в редакторе графики. В зависимости от её размера меняется масштаб и размер представления - холста. Кроме соответствующих кнопок, размер области редактирования(8x8 - 64x64) можно менять колесом прокрутки. При смещении рамки на спрайт-листе изменяется индекс спрайта - он отображается над областью редактирования графики. Выводимый индекс всегда соответствует самому левому-верхнему(первому) спрайту в выделенной области. Смещать область редактирования на спрайт-листе можно мышью либо стрелками на клавиатуре.

Инструменты и рабочая область графического редактора.
Над областью редактирования графики выводится индекс первого спрайта в выделенной области. Именно этот индекс передается в функцию вывода графики на экран `spr`. Ниже отображается область редактирования графики. Под ней мы видим кнопки инструментов: *"рисование"*, *"выделение"*, *"заливка"*, *"отразить по вертикальной оси"*, *"отразить по горизонтальной оси"*, *"повернуть"*, *"очистить"*.

- *"Рисование"* - стандартное рисование пикселями.
- *"Выделение"* - этот инструмент позволяет переместить выделенную область графики. Как только Вы активируете его, вокруг рабочей области появятся стрелки нажимая на которые происходит смещение выделенного участка на один пиксель. Выделенную область на холсте можно двигать стрелками на клавиатуре.
- *"Заливка"* - меняет цвет на текущий. Выберите нужный Вам цвет из 16 цветов текущей палитры. Кликнув на любой пиксель в области редактирования графики все пиксели его цвета поменяют цвет на новый.
- *"Отразить по вертикальной оси"* - отражение относительно вертикальной оси всей области редактирования.
- *"Отразить по горизонтальной оси"* - отражение относительно горизонтальной оси всей области редактирования.
- *"Повернуть"* - поворачивает всю область редактирования на 90 градусов по часовой стрелке.
- *"Очистить"* - очищает всю область редактирования заполняя её черным цветом.

В самом низу располагается цветовая палитра, состоящая из 16 цветов.
Левый клик на палитре выбирает первый(основной) цвет, правый клик - второй.
Соответственно, рисовать можно левой и правой кнопкой мыши. Можно выбрать основной цвет средним кликом на холсте.

Редактор карты тайлов

Редактор карты тайлов предназначен для расстановки спрайтов/тайлов на карте игрового мира.
Вся карта тайлов изначально заполнена "пустым" тайлом - спрайтом в спрайт-листе графики заднего плана с индексом 0.

Описание графического интерфейса

Блок кнопок в верху экрана, общий для всех встроенных редакторов.
В левом-верхнем углу экрана расположены кнопки перехода во встроенные редакторы: редактор кода, редактор спрайтов, редактор карты тайлов, редактор звуковых эффектов и редактор музыки.
Как и в любом другом встроенном редакторе **TIC-80**, Вы можете выполнить операции: вырезать, скопировать, вставить и отменить/вернуть изменения.
Эти действия могут быть выполнены как по нажатию на соответствующую кнопку в верхней части экрана, так и по 'горячим клавишам'.
Вверху по центру отображается текстовая подсказка для всех кнопок верхнего блока.

Интерфейс редактора карты тайлов.
Основное пространство экрана занимают ячейки для размещения тайлов 8x8 пикселей - их границы показывает мелкая сетка. Эту сетку можно отключить нажав на соответствующую кнопку в правом-верхнем углу экрана.
Также пространство разбито на бОльшие области имеющие размер экрана **TIC-80** - эта сетка отрисована более толстыми линиями и отображается всегда.
Перемещаться по карте можно перетягивая её мышью с зажатой правой кнопкой мыши(в веб-версии браузер перехватывает это нажатие и выполняет свои действия, например открывает предыдущую страницу), либо при помощи кнопок курсора. Карта тайлов зациклена - т.е. передвинув её левее первой ячейки можно попасть на самую последнюю, что позволяет быстро переместиться в противоположный край карты тайлов. Над курсором мыши отображаются координаты ячейки.
При наведении курсора на заполненную ячейку в верхней панели отобразится индекс тайла который в неё

установлен.
Карта тайлов имеет размер 240x136 ячеек 8x8 пикселей, что составляет в итоге 1920x1088 пикселей графики заднего плана. Нумерация ячеек начинается с нуля.

Кнопки в верхней-правой части:

- просмотр всей карты тайлов в уменьшенном виде, в котором можно выбрать следующий экран
- показать/скрыть сетку отображающую тайлы
- инструмент "рисования" тайлами
- инструмент "перетягивать карту"
- инструмент "выбор прямоугольной области"
- инструмент "заливка"
- показать/скрыть спрайт-лист графики заднего плана(BG) для выбора тайла который нужно расставить на карте.

В открытом окошке спрайт-листа графики заднего плана есть кнопка фиксации этого окошка, которая позволяет держать его постоянно открытым либо скрывать после выбора активного тайла.

Нажав на кнопку отображающую всю карту игрового мира - Вы увидите всю карту в уменьшенном виде, которая в свою очередь поделена на блоки имеющие размер экрана в уменьшенном виде. Выбрав любой из этих блоков Вы переместитесь на карте тайлов в указанный экран, который можно редактировать.

Редактор звуковых эффектов (v.0.22)

Редактор звуковых эффектов предназначен для создания и изменения звуковых эффектов.
Каждый из доступных 64 эффектов может быть проигран функцией API `sfx`.

Базовые знания

У нас есть 12 нот и 8 октав - всего получается 96 нот.
У каждой ноты своя частота и если не менять никаких настроек в редакторе, то будет просто играть чистая нота. Чтобы теперь получить звуковой эффект, надо как то менять частоту звучания и/или громкость со временем.
Для этого у нас есть следующие инструменты:

- **VOLUME**: изменяет громкость звучания
- **ARPEGGIO**: изменяет номер ноты относительно базовой ноты
- **PITCH**: изменяет частоту ноты относительно базовой частоты ноты (обычно используется для эффекта "дрожания" звука)
- **DUTY**: изменяет тембр звука (работает только для квадратной формы волны)

Описание графического интерфейса

Блок кнопок в верху экрана, общий для всех встроенных редакторов. В левом-верхнем углу экрана расположены кнопки перехода во встроенные редакторы: редактор кода, редактор спрайтов, редактор карты тайлов, редактор звуковых эффектов и редактор музыки. Как и в любом другом встроенном редакторе **TIC-80**, Вы можете выполнить операции: вырезать, скопировать, вставить и отменить/вернуть изменения.
Эти действия могут быть выполнены как по нажатию на соответствующую кнопку в верхней части экрана, так и по 'горячим клавишам'.
Вверху по центру отображается текстовая подсказка для всех кнопок верхнего блока.

Интерфейс редактора звуковых эффектов.
Сверху расположены кнопки: смена текущего звукового эффекта(0-63) и смена скорости проигрывания(восемь значений, от -4 до 3).
Центральное место занимает область редактирования одной из четырех характеристик эффекта: **VOLUME**, **ARPEGGIO**, **PITCH**, **DUTY**.
Значения меняются каждый кадр, т.е. у нас есть 30 значений - в поле редактирования соответствует количеству столбцов.
Проигрывание эффекта составляет 0.5 сек звучания на 0 скорости.
Каждое значение может быть от 0 до 15 - в поле редактирования это высота столбцов.
Для каждой характеристики эффекта можно установить параметры зацикливания **LOOP**:

- **SIZE**: размер зацикленной области 0-15
- **START**: индекс начала зацикленной области 0-15

Для **ARPEGGIO** имеется возможность изменять номер ноты вверх или вниз - кнопка **DOWN**
Для **PITCH** можно выставить изменение частоты ноты увеличенное в 16 раз - кнопка **x16**

Внизу отображены нотные клавиши охватывающие октаву.
Справа от клавиш расположены три кнопки - выбор формы сигнала:

- **SQR**: квадратная форма
- **TRI**: треугольная форма
- **NOISE**: форма шума

Под ними расположены 8 кнопок для выбора октавы.

Редактор музыки

Данная страница находится в стадии наполнения.

Сторонние редакторы

Встроенные редакторы призваны решать свою основную задачу - создавать и редактировать игру на любом устройстве. Они едины для всех платформ и всегда под рукой. Их разработка не останавливается и в них появляются всё новые "фишки", но что делать если Вы хотите использовать уже привычный для Вас профессиональный редактор? Такая разработка под **TIC-80** возможна!

Запуск TIC-80 с параметрами запуска

Нативные версии **TIC-80** (не web-версия) поддерживают некоторые аргументы при вызове из консоли или других программ.
Поддерживаются абсолютные и относительные пути. Полезно при использовании внешних редакторов.

```
tic game.tic ==> загружает картридж и запускает игру
tic -code game.lua ==> загружает и запускает код без начальной анимации
tic game.tic -code game.lua ==> загружает картридж, вставляет в него код и запускает игру без начальной анимации
tic game.tic -sprites image.gif ==> загружает картридж, вставляет в него спрайты и запускает игру без начальной анима
ции
tic cart.tic -map world.map ==> загружает картридж, вставляет в него данные карты тайлов и запускает игру без начальн
ой анимации
tic > log.txt ==> все сообщения отправленные в trace будут записаны в файл log.txt, который будет находиться рядом с
исполняемым файлом tic
tic | Out-File log.txt ==> также как и в предыдущем случае отправляет trace в файл log.txt, только для PowerShell
```

Использование стороннего редактора кода

Запуск TIC-80 с кодом во внешнем файле

Есть два варианта запуска **TIC-80** с кодом написанном в стороннем редакторе:

Первый вариант.
Разместите Ваш файл `game.lua` рядом с исполняемым файлом tic. Во встроенном редакторе кода наберите `dofile("game.lua")` в первой строке. Теперь изменив код в стороннем текстовом редакторе Вы можете перезапустить картридж TIC, например нажав комбинацию CTRL+R

Второй вариант.
Запуск TIC-80 с параметрами.
Выполнив команду в консоли операционной системы `tic -code game.lua` Вы запустите **TIC-80** в который будет встроен код из указанного Вами файла.
Редакторы кода и продвинутые редакторы текста позволяют подключить **TIC-80** для того чтобы код запускался в нем.
Например, для редактора **Sublime text** чтобы запустить код в **TIC-80** нужно проделать следующее:

- выбрать пункт меню Tools -> Build System -> New Build System
- написать в открывшемся файле следующее:

```
{
"cmd": ["путь_к_tic","-code","$file"],
"shell":true
}
```

  где `путь_к_tic` может быть, например `"C:\\Program Files\\TIC-80\\tic_0.21.0\\tic.exe"` - путь к исполняемому файлу **TIC-80**, для ОС Windows нужно указывать двойную косую черту

- сохранить файл указав имя `TIC-80`
- в меню Tools -> Build System выбрать `TIC-80` Теперь нажимая комбинацию клавиш `CTRL-B` либо выбрав соответствующий пункт меню, будет запускаться **TIC-80** в котором встроен Ваш код.
  Для запуска *конкретного картриджа* с Вашим кодом, необходимо прописать:

```
{
"cmd": ["путь_к_tic","путь_к_картриджу","-code","$file"],
"shell":true
}
```

  Не забываем, что пути для ОС Windows нужно указывать с двойной косой чертой.

Подсветка синтаксиса Lua/MoonScript

Язык программирования Lua является очень распространенным, поэтому подсветка его синтаксиса, как правило, входит в стандартный набор поддерживаемых языков редактора кода. Например, в редакторах **Notepad++** и **Sublime text** подсветка кода Lua включится автоматически, как только вы откроете файл с расширением `.lua`

Сниппеты и автодополнение кода

Рассмотрим редактор **Sublime text**.
Для Lua самые часто используемые сниппеты уже присутствуют: function, for, table. Для более полного охвата функций Вам понадобится установить отдельный пакет.

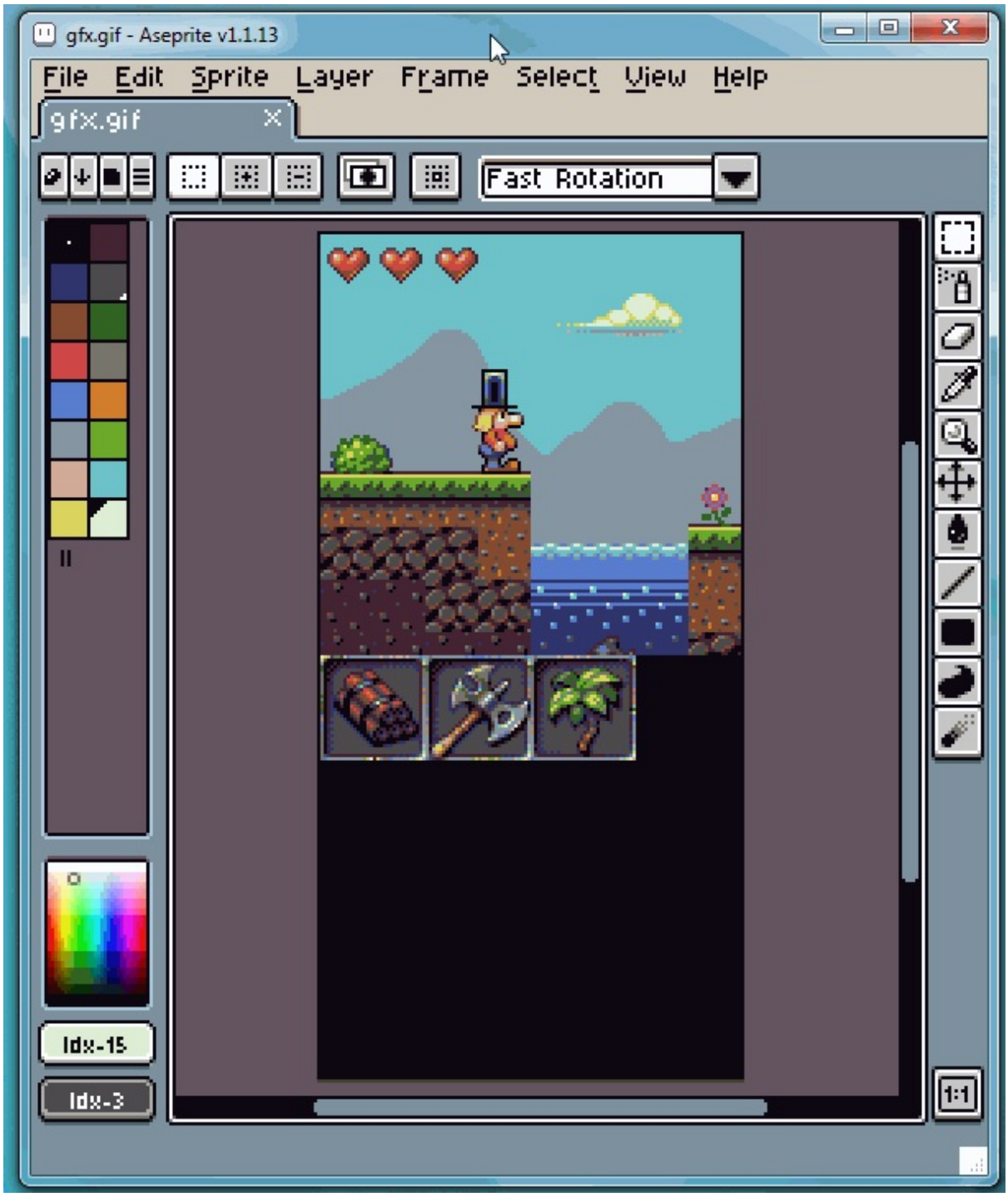
Специальный плагин для Sublime text 3

Существует пакет для Sublime Text 3 предназначенный для разработчиков игр под TIC-80. Он основан на пакетах BetterLua и Moonscripty. Пакет подсвечивает и автодополняет функции API TIC-80 и подключенные к нему стандартные библиотеки. Работает либо для Lua либо для MoonScript. Также позволяет создавать билд игры и запускать её в виртуальной игровой консоли TIC-80. Скачать его можно с github.

Специальный плагин для Visual Studio Code

Плагин для Visual Studio Code можно скачать отсюда

Использование стороннего редактора графики



Описание:

Для того чтобы подготовить графику для игры можно использовать сторонний редактор графики с последующим импортом полученного графического файла.

Перечислим необходимые шаги:

- выбор/создание палитры для использования в игре

- установка палитры в стороннем графическом редакторе
- рисование графики в стороннем графическом редакторе
- сохранение полученной графики в формате `.gif`, размером 128 пикселей в ширину и 256 пикселей в высоту
- установка выбранной палитры в картридже
- импорт графики в TIC-80 с помощью консольной команды `import sprites`

Примечание:
По-умолчанию в **TIC-80** используется палитра цветов `DB16`, она также имеется для многих пиксельных редакторов, например `Aseprite`.
Можно рисовать графику меньшим размером чем 128x256 пикселей, она импортируется в спрайт-лист графики заднего фона. Картинка 128x256(вертикальная картинка) импортируется в оба спрайт-листа: заднего фона(128x128) и переднего фона(128x128).

Использование стороннего редактора карты тайлов

Описание:

Для того чтобы подготовить карту тайлов для игры можно использовать сторонний редактор карты тайлов:

- TiledMapEditor-TIC-80 конвертер TiledMapEditor - TIC-80 (.tmx->.map, .map->.tmx) от Al Rado и Dark Hole
- Fantasy Console Map Tool редактор карт от MonsterGoBoom

Горячие клавиши

```
Сочетания клавиш, работающие в любом встроенном редакторе:
ESC                     переключение режимов консоль/редактор
ALT+~                   отображение консоли
ALT+1/F1                отображение редактора кода
ALT+2/F2                отображение редактора спрайтов
ALT+3/F3                отображение редактора карты тайлов
ALT+4/F4                отображение редактора звуковых эффектов
ALT+5/F5                отображение редактора музыки
F11/ALT+ENTER           полно-экранный/оконный режим
CTRL+R/ENTER            запуск текущего проекта
CTRL+X/C/V              вырезать/копировать/вставить
CTRL+Z/Y                отменить/вернуть изменения
CTRL+S                  сохранить картридж
F7                      назначить буфер экрана в качестве обложки
F8                      сохранить скриншот экрана TIC-80
F9                      сохранить экран TIC-80 в анимированный GIF

Редактор кода:
CTRL+F                  поиск текста
CTRL+G                  переход к номеру строки
CTRL+O                  отображение имён функций
CTRL+A                  выделить всё
CTRL+/                  закомментировать/раскомментировать строку
CTRL+HOME               переход в самый верх
CTRL+END                переход в самый низ
ARROWS                  навигация по коду
PAGE UP                 на страницу вверх
PAGE DOWN               на страницу вниз
MOUSE WHEEL             пролистать содержимое вверх/вниз
MOUSE LEFT BUTTON       выбрать позицию курсора
MOUSE RIGHT BUTTON      перетягивать содержимое вверх/вниз/влево/вправо (не работает в WEB)

Редактор спрайтов:
ARROWS                  передвинуть область отображения спрайтов
MOUSE WHEEL             уменьшить/увеличить область отображения спрайтов
MOUSE LEFT BUTTON       выбрать координату области отображения спрайтов, рисовать, выбрать цвет

Редактор карты тайлов:
ARROWS                  переместить карту тайлов клавишами
MOUSE RIGHT BUTTON      переместить карту тайлов мышью
SHIFT                   отображение спрайт-листа

Редактор звуковых эффектов:
1,2,3,4,5,6,7,8,9,0,-,= проиграть ноту текущим звуковым эффектом

Редактор музыки:
SHIFT+ENTER             проиграть паттерн с позиции курсора

Консоль:
UP                      отобразить предыдущую выполненную команду
DOWN                    отобразить следующую выполненную команду
LEFT                    переместить курсор влево
RIGHT                   переместить курсор вправо
TAB                     дополнить команду/имя файла
ENTER                   выполнить набранную команду
MOUSE WHEEL             пролистать содержимое вверх/вниз
```

Поддерживаемые платформы

TIC-80 написан на чистом языке C, с использованием библиотек SDL и LUA.
Поэтому он может запускаться на всех популярных платформах:

```
HMTL5               готово
Windows             готово
Windows 10 UWP      готово
Linux 32/64bit      готово
Android             готово
Mac OS X            готово
iOS                 готово
Raspberry Pi        готово
Pocket CHIP         готово
```

Вы можете найти iOS/tvOS версии здесь https://github.com/CliffsDover/TIC-80

Полный список функций API

Специальные функции

- **TIC** - главная функция обновления
- **scanline** - callback функция, вызываемая после рендера каждой строки
- **init** - пользовательская функция инициализации, вызываемая один раз перед выполнением кода

Опрос ввода/вывода

- **btn** - получение состояния кнопки джойстика в текущем кадре
- **btnp** - получение состояния кнопки джойстика в указанном периоде времени
- **mouse** - возвращает координаты и состояние нажатия мыши/касания

Вывод текста

- **print** - печать строки системным шрифтом
- **font** - печать строки шрифтом, отрисованном в карте спрайтов
- **trace** - печать строки в консоль

Вывод графики

- **spr** - печать спрайта
- **map** - печать области карты тайлов на экране
- **mset** - установка индекса тайла карты
- **mget** - получение индекса тайла карты
- **textri** - отображает треугольник, заполненный текстурой

Рисование

- **pix** - установка/получение цвета пикселя на экране
- **line** - рисование линии
- **circ** - рисование круга
- **circb** - рисование окружности
- **rect** - рисование заполненного прямоугольника
- **rectb** - рисование прямоугольника
- **tri** - рисование заполненного треугольника

Экран

- **cls** - очистка экрана
- **clip** - ограничение вывода на экран

Память

- **peek** - чтение байта из памяти
- **poke** - запись байта в память
- **peek4** - чтение полубайта из памяти
- **poke4** - запись полубайта в память
- **memcpy** - копирование байтов в памяти
- **memset** - заполнение памяти указанным значением

- **pmem** - запись целого значения в постоянную память (persistent cart data)

Звуки

- **sfx** - проигрывание звукового эффекта
- **music** - проигрывание музыки

Другое

- **time** - возвращает количество тиков с момента старта игры
- **sync** - сохранение изменений в спрайтах/карте тайлов во время игры
- **exit** - прерывает выполнение программы и производит выход в консоль

Специальные функции

Такие функции, как правило, представляют из себя callback функции - т.е. функции, которые вызываются системой.

TIC

главная функция

`TIC` это главная функция. Она вызывается с частотой 60 fps (60 раз в секунду).
У неё нет параметров и она является местом, в котором и совершается вся магия.
Это единственная функция, которая обязательно должна присутствовать в коде:

```lua
-- script: lua

function TIC()
  -- Ваш код здесь
end
```

```moon
-- script: moon

export TIC=->
  -- Ваш код здесь
```

```js
// script: js

function TIC() {
  // Ваш код здесь
}
```

scanline

callback функция, вызываемая после рендера каждой строки

scanline (`line` **)**

Параметры:

`line` - номер строки

Описание:

`scanline` это callback функция, как и главная функция `TIC`, но вызывается системой после рендера каждой СТРОКИ.

Идея в том, что можно менять палитру в этот момент. Т.е. имея 136 строк * 16 цветов получим 2176 цветов, которые можно отобразить за один кадр. Этот трюк называется "scanline trick" и использовался раньше на приставках, для увеличения количества цветов. Если Вы хотите почувствовать себя хакером - эта функция для Вас.

init

пользовательская функция инициализации, вызываемая один раз перед выполнением кода

По дизайну в TIC-80 имеется только одна обязательная функция ядра. Это функция `TIC`.
Однако, многие пользователи хотели бы иметь функцию инициализации, вызываемую один раз перед выполнением основного кода.

Пример:

Запустить или скачать картридж примера.

```
-- title:  init demo
-- author:
-- desc:
-- script: lua
-- input:  gamepad

function init()
  msg="inited"
  t=0
end

init()

function TIC()
  cls(0)
  print("status: "..msg.." "..t)
  t=t+1
end
```

Запустить или скачать картридж примера.

```
-- title:  init demo
-- author:
```

```
-- desc:
-- script: moon
-- input:  gamepad

local msg
local t

INIT=->
  msg="inited"
  t=0

INIT()

export TIC=->
  cls 0
  print "status: "..msg.." "..t
  t+=1
```

```
// title:  init demo
// author:
// desc:
// script: js
// input:  gamepad

function init() {
  msg = "inited"
  t = 0
}

init()

function TIC() {
  cls(0)
  print("status: " + msg + " " + t)
  t++
}
```

Опрос ввода/вывода

Функции API, обеспечивающие взаимодействие игрока и игровой системы.

btn

получение состояния кнопки джойстика в текущем кадре

btn (`[id]` **) ->** `pressed`

Параметры:

`id` - код кнопки, которую нужно опросить. Коды кнопок: первый джойстик `0..5`, второй джойстик `8..13`

Возвращает:

`pressed` - состояние опрашиваемой кнопки в текущем кадре, `true` если кнопка нажата.

Описание:

Функция опрашивает состояние кнопки подключенной к TIC.

Коды кнопок джойстиков:

Первый джойстик	Код	Второй джойстик	Код
Вверх:	**0**	Вверх:	**8**
Вниз:	**1**	Вниз:	**9**
Влево:	**2**	Влево:	**10**
Вправо:	**3**	Вправо:	**11**
Кнопка А:	**4**	Кнопка А:	**12**
Кнопка Б:	**5**	Кнопка Б:	**13**

Пример:

Запустить или скачать картридж примера.

```
-- script: lua
-- title:  btn demo
-- author: Rain_Effect
```

```
local x=70
local y=25
local btnLabel={"Up","Down","Left","Right","Btn A","Btn B"}

function TIC()
  cls(0)
  print("Key Test Project",x,y)
  print("Btn",x,y+15,2)
  print("1P",x+50,y+15,2)
  print("2P",x+80,y+15,2)
  for i=1,6 do
    print(btnLabel[i],x,y+(i+1)*10+5)
    if btn(i-1) then
      print("On",x+50,y+(i+1)*10+5,11)
    else
     print("Off",x+47,y+(i+1)*10+5,6)
    end
    if btn(i+7) then
      print("On",x+80,y+(i+1)*10+5,11)
    else
      print("Off",x+77,y+(i+1)*10+5,6)
    end
  end
end
```

Запустить или скачать картридж примера.

```
-- script: moon
-- title:  btn demo
-- author: Rain_Effect

x=70
y=25
btnLabel={"Up","Down","Left","Right","Btn A","Btn B"}

export TIC=->
  cls 0
  print "Key Test Project",x,y
  print "Btn",x,y+15,2
  print "1P",x+50,y+15,2
  print "2P",x+80,y+15,2
  for i=1,6
    print btnLabel[i],x,y+(i+1)*10+5
    if btn(i-1)
      print "On",x+50,y+(i+1)*10+5,11
    else
      print "Off",x+47,y+(i+1)*10+5,6
    if btn(i+7)
      print "On",x+80,y+(i+1)*10+5,11
    else
      print "Off",x+77,y+(i+1)*10+5,6
```

```
// script: js
// title:  btn demo
// author: Rain_Effect

var x = 70
var y = 25
var btnLabel = ["Up", "Down", "Left", "Right", "Btn A", "Btn B"]

function TIC() {
  cls(0)
```

```
  print("Key Test Project", x, y)
  print("Btn", x, y + 15, 2)
  print("1P", x + 50, y + 15, 2)
  print("2P", x + 80, y + 15, 2)
  for (var i = 0; i < 6; ++i) {
    print(btnLabel[i], x, y + (i + 2) * 10 + 5)
    if (btn(i))
      print("On", x + 50, y + (i + 2) * 10 + 5, 11)
    else
      print("Off", x + 47, y + (i + 2) * 10 + 5, 6)
    if (btn(i + 8))
      print("On", x + 80, y + (i + 2) * 10 + 5, 11)
    else
      print("Off", x + 77, y + (i + 2) * 10 + 5, 6)
  }
}
```

btnp

получение состояния кнопки джойстика в указанном периоде времени

btnp (`[id, [hold, period]]` **)** -> `pressed`

Параметры:

`id` - код кнопки, которую нужно опросить. Коды кнопок смотрите в описании

`hold` - время (в тиках) сколько кнопка должна быть нажата, чтобы рассматривать её состояние как "нажата"

`period` - время (в тиках), после которого функция будет возвращать `true` снова

Возвращает:

`pressed` - состояние опрашиваемой кнопки, `true` если кнопка нажата

Описание:

Эта функция позволяет читать статус одной из кнопок, задействованных в TIC.
Функция возвращает значение `true` только в момент нажатия на клавишу.
Она также может быть использована с параметрами `hold` и `period`, которые позволяют возвращать `true` во время нажатия кнопки. После того как время нажатия `hold` закончится, функция вернет true каждый раз когда закончится время указанного периода `period`.
Время выражается в тиках: при 60 fps это означает, что 120 тиков равны 2 секундам.

Коды кнопок джойстиков:

Первый джойстик	Код	Второй джойстик	Код
Вверх:	**0**	Вверх:	**8**
Вниз:	**1**	Вниз:	**9**
Влево:	**2**	Влево:	**10**
Вправо:	**3**	Вправо:	**11**
Кнопка А:	**4**	Кнопка А:	**12**
Кнопка Б:	**5**	Кнопка Б:	**13**

Пример:

Запустить или скачать картридж примера.

```
-- title:  btnp demo
-- author:
-- desc:   move the rectangle in 10 pixels step,
-- every time one direction keys is pressed.
-- the rectangle move every tenth of seconds.
-- script: lua
-- input:  gamepad

x=120
y=80

cls(12)
function TIC()
  if btnp(0,60,6) then y=y-10 end
  if btnp(1,60,6) then y=y+10 end
  if btnp(2,60,6) then x=x-10 end
  if btnp(3,60,6) then x=x+10 end

  rect(x,y,10,10,8)
end
```

Запустить или скачать картридж примера.

```
-- title:  btnp demo
-- author:
-- desc:   move the rectangle in 10 pixels step,
-- every time one direction keys is pressed.
-- the rectangle move every tenth of seconds.
-- script: moon
-- input:  gamepad

x=120
y=80

cls 12
export TIC=->
  if btnp(0,60,6) then y-=10
  if btnp(1,60,6) then y+=10
```

```
  if btnp(2,60,6) then x-=10
  if btnp(3,60,6) then x+=10

  rect x,y,10,10,8
```

```
// title:  btnp demo
// author:
// desc:   move the rectangle in 10 pixels step,
// every time one direction keys is pressed.
// the rectangle move every tenth of seconds.
// script: js
// input:  gamepad

var x = 120
var y = 80

cls(12)
function TIC() {
  if (btnp(0, 60, 6)) y -= 10
  if (btnp(1, 60, 6)) y += 10
  if (btnp(2, 60, 6)) x -= 10
  if (btnp(3, 60, 6)) x += 10

  rect(x, y, 10, 10, 8)
}
```

mouse

возвращает координаты и состояние нажатия мыши/касания

mouse() -> `x, y, pressed`

Описание: Функция возвращает координаты мыши и состояние нажатия.
Для использования этой функции в метаданных картриджа должно быть прописано `-- input: mouse`.
Будьте внимательны, включение поддержки мыши отключит джойстики.

Пример:

Запустить или скачать картридж примера.

```
-- title:  mouse demo
-- author: Raidez
-- desc:
-- script: lua
-- input:  mouse

t=0
x=104
y=24

function TIC()
  mx,my,md=mouse() --get x,y and pressed

  if md then
    x=mx
    y=my
  end

  cls(12)
  spr(1+(t%60)/30,x,y,-1,4)
  t=t+1
end
```

Запустить или скачать картридж примера.

```
-- title:  mouse demo
-- author: Raidez
-- desc:
-- script: moon
-- input:  mouse

t=0
x=104
y=24

export TIC=->
  mx,my,md=mouse() --get x,y and pressed

  if md
    x=mx
    y=my

  cls 12
  spr 1+(t%60)/30,x,y,-1,4
  t=t+1
```

```
// title:  mouse demo
// author: Raidez
// desc:
// script: js
// input:  mouse

var t = 0
var x = 104
var y = 24

function TIC() {
  var m = mouse()
  var mx = m[0] //get x,y and pressed
  var my = m[1]
  var md = m[2]

  if (md) {
    x = mx
    y = my
  }

  cls(12)
  spr(1 + (t % 60) / 30, x, y, -1, 4)
  t++
}
```

Пример:

Запустить или скачать картридж примера.

```
-- title:  mouse demo
-- author: Filippo
-- desc:
-- script: lua
-- input:  mouse
-- pal:    DB16

r=0
function TIC()
  cls(0)

  --get mouse info
  x,y,p=mouse()

  --if pressed
  if p then r=r+2 end r=r-1
  r=math.max(0, math.min(32, r))

  --draw stuff
  line(x,0,x,136,11)
  line(0,y,240,y,11)
  circ(x,y,r,11)

  --show coordinates
  c=string.format('(%03i,%03i)',x,y)
  print(c,0,0,15,1)
end
```

Запустить или скачать картридж примера.

```
-- title:  mouse demo
-- author: Filippo
-- desc:
-- script: moon
-- input:  mouse
-- pal:    DB16

r=0
```

```
export TIC=->
  cls 0

  --get mouse info
  x,y,p=mouse()

  --if pressed
  if p then r=r+2
  r-=1
  r=math.max(0, math.min(32, r))

  --draw stuff
  line x,0,x,136,11
  line 0,y,240,y,11
  circ x,y,r,11

  --show coordinates
  c=string.format '(%03i,%03i)',x,y
  print c,0,0,15,1
```

```
// title:  mouse demo
// author: Filippo
// desc:
// script: js
// input:  mouse

r = 0
function TIC() {
  cls(0)

  //get mouse info
  var m = mouse()
  var x = m[0]
  var y = m[1]
  var p = m[2]

  //if pressed
  if (p) r = r + 2
  r = r - 1
  r = Math.max(0, Math.min(32, r))

  //draw stuff
  line(x, 0, x, 136, 11)
  line(0, y, 240, y, 11)
  circ(x, y, r, 11)

  //show coordinates
  c = "(" + x + "," + y + ")"
  print(c, 0, 0, 15, 1)
}
```

Вывод текста

Функции позволяющие выводить текст/значения переменных на экран либо консоль.

print

печать строки системным шрифтом

print (`text, [x=0, y=0, [color=15, [fixed=false, [scale=1]]]]` **) ->** `width`

Параметры:

`text` - строка для вывода на экран

`x` - координата на экране по оси x

`y` - координата на экране по оси y

`color` - цвет текста

`fixed` - флаг, указывающий фиксировать ли ширину символов

`scale` - масштаб шрифта, целое число

Возвращает:

`width` - ширина напечатанного текста в пикселях

Описание:

Просто печатает текст на экран, используя системный шрифт, заданный в файле конфигурации.
Может печатать многострочный текст - для переноса строки используйте \n.

Для печати специально заданным шрифтом, воспользуйтесь оператором font.
Для печати в консоль воспользуйтесь оператором trace.

Пример:

Запустить или скачать картридж примера.

```
-- title:  print demo
-- author: Filippo
-- desc:   print matrix
-- script: lua
-- input:  gamepad

msg="FNORD                    "
t=0
```

```
function TIC()
  cls()
  c=1
  for x=0,29 do
    for y=0,16 do
      c=(c+1)%#msg
      l=(c-math.floor(t))%#msg
      print(msg:sub(l,l),x*8,y*8,y%12)
    end
  end
  t=t+0.15
end
```

Запустить или скачать картридж примера.

```
-- title:  print demo
-- author: Filippo
-- desc:   print matrix
-- script: moon
-- input:  gamepad

msg="FNORD                      "
t=0
export TIC=->
  cls()
  c=1
  for x=0,29
    for y=0,16
      c=(c+1)%#msg
      l=(c-math.floor(t))%#msg
      print msg\sub(l,l),x*8,y*8,y%12
  t+=0.15
```

```
// title:  print demo
// author: Filippo
// desc:   print matrix
// script: js
// input:  gamepad

var msg = "FNORD                      "
var t = 0
function TIC() {
  cls()
  var c = 1
  for (var x = 0; x < 30; ++x) {
    for (var y = 0; y < 17; ++y) {
      c = (c + 1) % msg.length
      l = (c - Math.floor(t))
      l = l - Math.floor(l / msg.length) * msg.length
      print(msg[l], x * 8, y * 8, y % 12)
    }
  }
  t += 0.15
}
```

font

печать строки шрифтом, отрисованном в карте спрайтов

font (`text, [x=0, y=0, [alpha_color=15, [w=8, h=8, [fixed=false, [scale=1]]]]]` **) ->** `width`

Параметры:

`text` - строка для вывода на экран

`x` - координата на экране по оси x

`y` - координата на экране по оси y

`alpha_color` - индекс прозрачного цвета

`w` - ширина пустого символа(например пробела) в пикселях

`h` - высота пустого символа(например пробела) в пикселях

`fixed` - флаг, указывающий фиксировать ли ширину символов

`scale` - масштаб шрифта, целое число

Возвращает:

`width` - ширина напечатанного текста в пикселях

Описание:

Печатает текст на экран, используя пользовательский шрифт, заданный в области спрайтов переднего плана (FG).

Может печатать многострочный текст - для переноса строки используйте \n.

Пример:

Примечание:

Для корректной работы данного примера Вам понадобится картридж.

Запустить или скачать картридж примера.

```
-- title:  font
-- author: Nesbox
-- desc:   example how to render font from the foreground sprites
-- script: lua
```

```
-- input:  gamepad

cls(12)
local text="The quick brown fox\njumps over the lazy dog!"

font(text,8,8,5)
font(text,8,32,5,8,8,true)
font(text,8,58,5,5,8,false,2)

function TIC()end
```

Запустить или скачать картридж примера.

```
-- title:  font
-- author: Nesbox
-- desc:   example how to render font from the foreground sprites
-- script: moon
-- input:  gamepad

cls 12
text="The quick brown fox\njumps over the lazy dog!"

font(text,8,8,5)
font(text,8,32,5,8,8,true)
font(text,8,58,5,5,8,false,2)

export TIC=->
```

```
// title:  font
// author: Nesbox
// desc:   example how to render font from the foreground sprites
// script: js
// input:  gamepad

cls(12)
var text = "The quick brown fox\njumps over the lazy dog!"

font(text, 8, 8, 5)
font(text, 8, 32, 5, 8, 8, true)
font(text, 8, 58, 5, 5, 8, false, 2)

function TIC() { }
```

trace

печать строки в консоль

trace (`msg, [color=15]` **)**

Параметры:

`msg` - сообщение для вывода в консоль. Может быть строкой или переменной

`color` - цвет выводимого сообщения

Описание:

Это сервисная функция для отладки кода. Выводит в консоль переданный параметр.

Советы:

1. Для объединения строк в Lua/MoonScript используйте `..`
2. Для очистки консоли используйте команду консоли `cls`

Пример:

Запустить или скачать картридж примера.

```
-- title:  demo
-- author:
-- desc:
-- script: lua
-- input:  gamepad

cls(0)

function TIC()
  trace('hello console:'..time())
end
```

Запустить или скачать картридж примера.

```
-- title:  demo
-- author:
-- desc:
-- script: moon
-- input:  gamepad

cls 0

export TIC=->
  trace 'hello console: '..time()
```

```
// title:  demo
// author:
// desc:
// script: js
// input:  gamepad

cls(0)

function TIC() {
  trace("hello console:" + time())
}
```

Вывод графики

Функции для вывода на экран и изменения спрайтов/тайлов фона.

spr

печать спрайта

spr (`id, x, y, [alpha_color=-1, [scale=1, [flip=0, [rotate=0, [cell_width=1, cell_height=1]]]]]` **)**

Параметры:
`id` - индекс спрайта (0-511) `x` - координата по оси x, по которой будет отрисован спрайт, начало координат левый-верхний угол экрана
`y` - координата по оси y, по которой будет отрисован спрайт, начало координат левый-верхний угол экрана
`alpha_color` - индекс цвета, который будет прозрачным. Используйте `-1` для отсутствия прозрачных пикселей, также *может быть массивом с индексами прозрачных цветов*
`scale` - масштаб, который будет применен к спрайту
`flip` - отражение спрайта по вертикальной/горизонтальной оси, либо по обеим осям сразу
`rotate` - поворот спрайта на 0, 90, 180 или 270 градусов
`cell_width` - ширина, измеряется в ячейках
`cell_height` - высота, измеряется в ячейках

Описание:
Выводит на экран спрайт по указанному индексу в определенные координаты.
Вы можете выбрать цвет из палитры, который будет прозрачным. Используйте `-1` для отсутствия прозрачных пикселей

Спрайт может быть отмасштабирован по указанному коэффициенту. Например, при масштабе равном `2`, спрайт 8x8 будет выведен на экран размером 16x16 пикселей.

Вы можете отразить спрайт передавая параметр `flip`:
`0` - нет отражения
`1` - отражение относительно вертикальной оси
`2` - отражение относительно горизонтальной оси
`3` - отражение по обеим осям сразу

Когда вы поворачиваете спрайт, передавая параметр `rotate`, он поворачивается с шагом в 90° по часовой стрелке:
`0` - нет поворота
`1` - поворот на 90°
`2` - поворот на 180°
`3` - поворот на 270°

Пример:

Запустить или скачать картридж примера.

```
-- title:  spr demo
-- author:
-- desc:
-- script: lua
-- input:  gamepad

--Build sprite from text
data="0100000001600100152401601701701701701701751701662502551701701701751061701701701701700510510510510"
for i=0,31 do
  poke(0x4000+32+i,
  tonumber(string.sub(data,i*3+1,i*3+3)))
end

p={
  x={val=100,min=0,max=240},
  y={val=68,min=0,max=136},
  colorkey={val=0,min=-1,max=15},
  scale={val=1,min=0,max=256},
  flip={val=0,min=0,max=3},
  rotate={val=0,min=0,max=3},
}
sel=next(p,nil)

function TIC()
  --Keys
  if btnp(0,30,6) then
    for n=0,4 do
      sel=next(p,sel) or next(p,nil)
    end
  end
  if btnp(1,30,6) then
    sel=next(p,sel) or next(p,nil)
  end

  if btnp(2,30,6) then
    p[sel].val=p[sel].val-1
  end
  if btnp(3,30,6) then
    p[sel].val=p[sel].val+1
```

```
  end

  --Clamp
  if p[sel].val>p[sel].max then
    p[sel].val=p[sel].max
  end
  if p[sel].val<p[sel].min then
    p[sel].val=p[sel].min
  end

  cls(0)
  print("Use up/down  to select parameter",0,0)
  print("left/right to change its value:",0,10)

  --Build menu with cursor
  r=0
  for k,v in pairs(p) do
    cur=(k==sel) and '>' or ' '
    print(cur..k..':'..v.val,0,30+10*r)
    r=r+1
  end

  --Draw Sprite
  spr(1,
      p.x.val,
      p.y.val,
      p.colorkey.val,
      p.scale.val,
      p.flip.val,
      p.rotate.val)

end
```

Запустить или скачать картридж примера.

```
-- title:  spr demo
-- author:
-- desc:
-- script: moon
-- input:  gamepad

--Build sprite from text
data="0100000001600100152401601701701701701701751701662502551701701701751061701701701701700510510510510510"
for i=0,31
  poke 0x4000+32+i, data\sub(i*3+1,i*3+3)

p={
  x:{val:100,min:0,max:240}
  y:{val:68,min:0,max:136}
  colorkey:{val:0,min:-1,max:15}
  scale:{val:1,min:0,max:256}
  flip:{val:0,min:0,max:3}
  rotate:{val:0,min:0,max:3}
}
sel=next(p,nil)

export TIC=->
  --Keys
  if btnp 0,30,6
    for n=0,4 do sel=next(p,sel) or next(p,nil)
  if btnp 1,30,6 then sel=next(p,sel) or next(p,nil)
  if btnp 2,30,6 then p[sel].val=p[sel].val-1
  if btnp 3,30,6 then p[sel].val=p[sel].val+1
```

```
  --Clamp
  if p[sel].val>p[sel].max then p[sel].val=p[sel].max
  if p[sel].val<p[sel].min then p[sel].val=p[sel].min

  cls 0
  print "Use up/down  to select parameter",0,0
  print "left/right to change its value:",0,10

  --Build menu with cursor
  r=0
  for k,v in pairs(p)
    cur=(k==sel) and '>' or ' '
    print cur..k..':'..v.val,0,30+10*r
    r+=1

 --Draw Sprite
  spr(1,
    p.x.val,
    p.y.val,
    p.colorkey.val,
    p.scale.val,
    p.flip.val,
    p.rotate.val)
```

```
// title:  spr demo / lua
// author: Filippo
// desc:   shows how to use 'spr'
// script: js
// input:  gamepad

//Build sprite from text
data = "010000000160010015240160170170170170170175170166250255170170170175106170170170170170051051051051"
for (var i = 0; i < 32; ++i) {
  poke(0xf4000 + 32 + i,
    parseInt(data.slice(i * 3 + 1, i * 3 + 3)))
}

var p = {
  x: { val: 100, min: 0, max: 240 },
  y: { val: 68,  min: 0, max: 136 },
  colorkey: { val: 0, min: -1, max: 15 },
  scale: { val: 1, min: 0, max: 256 },
  flip: { val: 0, min: 0, max: 3 },
  rotate: { val: 0, min: 0, max: 3 }
}
function next(o, i) {
  var ks = Object.keys(o)
  if (i == undefined || i == null)
    return ks[0];
  var i = ks.indexOf(i) + 1
  return i > ks.length ? "" : ks[i]
}
var sel = next(p, null)

function TIC() {
  //Keys
  if (btnp(0, 30, 6)) {
    for (var n = 0; n < 5; ++n)
      sel = next(p, sel) || next(p, null)
  }
  if (btnp(1, 30, 6)) {
    sel = next(p, sel) || next(p, null)
  }

  if (btnp(2, 30, 6)) {
```

```
    p[sel].val = p[sel].val - 1
  }
  if (btnp(3, 30, 6)) {
    p[sel].val = p[sel].val + 1
  }

  //Clamp
  if (p[sel].val > p[sel].max) {
    p[sel].val = p[sel].max
  }
  if (p[sel].val < p[sel].min) {
    p[sel].val = p[sel].min
  }

  cls(0)
  print("Use up/down  to select parameter", 0, 0)
  print("left/right to change its value:", 0, 10)

  //Build menu with cursor
  var r = 0
  for (var k in p) {
    var v = p[k]
    cur = (k == sel) ? ">" : " "
    print(cur + k + ":" + v.val, 0, 30 + 10 * r)
    r = r + 1
  }

  //Draw Sprite
  spr(1,
    p.x.val,
    p.y.val,
    p.colorkey.val,
    p.scale.val,
    p.flip.val,
    p.rotate.val)
}
```

map

печать области карты тайлов на экране

map (`[cell_x=0, cell_y=0, [cell_w=30, cell_h=17, [x=0, y=0, [alpha_color=-1, [scale=1 ,[remap=nil]]]]]]` **)**

Параметры:

`cell_x` - самая левая ячейка, для вывода карты

`cell_y` - самая верхняя ячейка, для вывода карты

`cell_w` - количество ячеек по горизонтали, начиная с cellx

`cell_h` - количество ячеек по вертикали, начиная с celly

`x` - координата по оси x для вывода карты на экран

`y` - координата по оси y для вывода карты на экран

`alpha_color` - индексированный цвет (0-15) который будет прозрачным. По умолчанию, прозрачность не задана

`scale` - масштаб, целое число

`remap` - callback функция, позволяющая показывать/скрывать/отражать/поворачивать тайлы во время вывода области карты на экран

Сигнатура функции remap:

remap (`[tile, [x, [y]]]` **)** -> `[tile, [flip, [rotate]]]`

Описание:

Карта измеряется в ячейках, блоки 8x8 пикселей, куда вы можете поставить тайл в редакторе карты тайлов. Функция может печатать всю карту либо часть её. Максимальный размер карты ограничен значением 240x136 ячеек.

Пример:

Запустить или скачать картридж примера.

```
-- title:  remap demo
-- author: nesbox
-- desc:   shows how to use map callback
```

```
-- script: lua
-- input:  gamepad

t=0
x=104
y=24

function TIC()
  cls(13)
  map(0,0,30,17,0,0,0,1,
    function(tile)
      return tile~=0 and tile+(2*(t%30//10))
    end)
  t=t+1
end
```

Запустить или скачать картридж примера.

```
-- title:  remap demo
-- author: nesbox
-- desc:   shows how to use map callback
-- script: moon
-- input:  gamepad

t=0
x=104
y=24

export TIC=->
  cls 13
  map 0,0,30,17,0,0,0,1,
    (tile)->
      return tile~=0 and tile+(2*(t%30//10))
  t=t+1
```

```
// title: remap demo
// author: nesbox
// desc: shows how to use map callback
// script: js
// input: gamepad

t = 0
x = 104
y = 24

function TIC() {
  cls(13)
  map(0, 0, 30, 17, 0, 0, 0, 1,
    function (tile) {
      return tile!= 0 && tile + (2 * (Math.floor(t % 30 / 10)))
    })
  t++
}
```

mset

установка индекса тайла карты

mset (`cell_x, cell_y, index` **)**

Параметры:

`cell_x` - координата на карте тайлов по оси x

`cell_y` - координата на карте тайлов по оси y

`index` - индекс тайла (0-255)

Описание:

Устанавливает индекс тайла карты, т.е. индекс спрайта, который размещен в спрайт-листе графики заднего плана, индексы 0-255.

По умолчанию внесенные изменения сохраняются только во время текущей игры. Чтобы внести постоянные изменения в карту, см. sync

Совет:

Память карты тайлов можно использовать по своему усмотрению и записывать туда любую информацию. Размер карты 240x136 тайлов, индекс тайла занимает 1 байт. Таким образом, карта занимает в памяти 240*136=32640 байт.

Пример:

Запустить или скачать картридж примера.

```
-- title:  mset demo
-- author:
-- desc:
-- script: lua
-- input:  gamepad

local sw, sh = 240, 136

function ClearMap()
  for x = 0,sw-1 do
```

```
    for y = 0, sh-1 do
      mset(x,y,0)
    end
  end
end

function MakeRoom()
  local size=5
  for x = 0, size do
    for y = 0, size do
      if x == 0 or y == 0 or x == size or y == size then
        mset(x,y,1)
      end
    end
  end
end

ClearMap()
MakeRoom()
sync()

function TIC()
  cls()
  map()
end
```

Запустить или скачать картридж примера.

```
-- title:  mset demo
-- author:
-- desc:
-- script: moon
-- input:  gamepad

sw, sh = 240, 136

ClearMap=->
  for x = 0,sw-1
    for y = 0, sh-1
      mset(x,y,0)

MakeRoom=->
  size=5
  for x = 0, size
    for y = 0, size
      if x == 0 or y == 0 or x == size or y == size
        mset(x,y,1)

ClearMap()
MakeRoom()
sync()

export TIC=->
  cls()
  map()
```

```
// title:  mset demo
// author:
// desc:
// script: js
// input:  gamepad

var sw=240, sh = 136
```

```
function ClearMap() {
  for (x = 0; x < sw - 1; x++) {
    for (y = 0; y < sh - 1; y++) {
      mset(x, y, 0)
    }
  }
}

function MakeRoom() {
  var size = 5
  for (x = 0; x < size+1; x++) {
    for (y = 0; y < size+1; y++) {
      if (x == 0 || y == 0 || x == size || y == size) {
        mset(x, y, 1)
      }
    }
  }
}

ClearMap()
MakeRoom()
sync()

function TIC() {
  cls()
  map()
}
```

mget

получение индекса тайла карты

mget (`cell_x, cell_y` **) ->** `index`

Параметры:

`cell_x` - координата на карте тайлов по оси x

`cell_y` - координата на карте тайлов по оси y

Возвращает:

`index` - индекс тайла (0-255)

Описание:

Считывает индекс тайла по указанным координатам на карте тайлов. Этот индекс соответствует индексу спрайта, который размещен в спрайт-листе графики заднего плана, индексы 0-255.

textri

отображает треугольник, заполненный текстурой

textri (`x1, y1, x2, y2, x3, y3, u1, v1, u2, v2, u3, v3, [use_map=false, [alpha_color=-1]]` **)**

Параметры:

`x1` - координата по оси x, первого угла треугольника

`y1` - координата по оси y, первого угла треугольника

`x2` - координата по оси x, второго угла треугольника

`y2` - координата по оси y, второго угла треугольника

`x3` - координата по оси x, третьего угла треугольника

`y3` - координата по оси y, третьего угла треугольника

UV координаты:

Это можно рассматривать как окно внутри изображения, или карту.

`u1` : координата U, первого угла треугольника

`v1` : координата V, первого угла треугольника

`u2` : координата U, второго угла треугольника

`v2` : координата V, второго угла треугольника

`u3` : координата U, третьего угла треугольника

`v3` : координата V, третьего угла треугольника

`use_map` : использовать ли вывод видео-памяти напрямую, или использовать память карты тайлов

`alpha_color` : индекс прозрачного цвета

Описание:

Отображает треугольник, заполненный текстурой с помощью изображения ram или map ram.

Рисование

Функции для рисования на экране геометрических фигур.

pix

установка/получение цвета пикселя на экране

pix (`x, y, [color]` **) ->** `color`

Параметры:

`x` - координата по оси x, где находится пиксель

`y` - координата по оси y, где находится пиксель

`color` - индекс цвета в палитре, для печати по указанным координатам

Возвращает:

`color` - возвращает индекс цвета в палитре цветов (0-15), который находится по указанным координатам

Описание:

Функция рисует цветной пиксель по указанным координатам.

Также может использоваться только для получения значения цвета пикселя на экране.

Пример:

Запустить или скачать картридж примера.

```
-- title:  pix demo
-- author:
-- desc:   put color
-- script: lua
-- input:  gamepad

cls()
function TIC()
  for i=0,6000 do
    x=math.random(240)
    y=math.random(136)
    --Put a math colored pixel at random place
    pix(x,y,(time()//1000*x*y)%60)
  end
end
```

Пример:

Запустить или скачать картридж примера.

```
-- title:  pix demo
-- author:
-- desc:   read a color
-- script: lua
-- input:  gamepad

t=0
--Draw some background
cls(0)
for i=0,15 do
 rect(9*i,6*i,6*i,3*i,i)
end

function TIC()
  if(t>12)then --wait some time
    t=0
    for x=0,240,2 do   --every 2 pixel in width
      for y=0,136,2 do  --every 2 pixel in height
        c=pix(x,y)       --take color
        c=(c+1)%15       --change it
        pix(x,y,c)       --put it back
      end
    end
  end
  t=t+1
end
```

Запустить или скачать картридж примера.

```
-- title:  pix demo
-- author:
-- desc:   put color
-- script: moon
-- input:  gamepad

cls()
export TIC=->
  for i=0,6000
```

```
    x=math.random(240)
    y=math.random(136)
    --Put a math colored pixel at random place
    pix x,y,(time()//1000*x*y)%60
```

Пример:

Запустить или скачать картридж примера.

```
-- title:  pix demo
-- author:
-- desc:   read a color
-- script: moon
-- input:  gamepad

t=0
--Draw some background
cls 0
for i=0,15
  rect 9*i,6*i,6*i,3*i,i

export TIC=->
  if t>12 --wait some time
    t=0
    for x=0,240,2 do   --every 2 pixel in width
      for y=0,136,2 do  --every 2 pixel in height
        c=pix(x,y)       --take color
        c=(c+1)%15       --change it
        pix x,y,c        --put it back
  t=t+1
```

```
// title: pix demo
// author:
// desc: put color
// script: js
// input: gamepad

cls()

function TIC() {
  for (i = 0; i < 6000; i++) {
```

```
    x = Math.random() * 240
    y = Math.random() * 136
    //Put a math colored pixel at random place
    pix(x, y, (time() / 1000 * x * y) % 60)
  }
}
```

Пример:


```
// title: pix demo
// author:
// desc: read a color
// script: js
// input: gamepad

t = 0
//Draw some background
cls(0)
for (i = 0; i < 15; i++) {
  rect(9 * i, 6 * i, 6 * i, 3 * i, i)
}

function TIC() {
  if (t > 12) { //wait some time
    t = 0
    for (x = 0; x < 240; x += 2) { //every 2 pixel in width
      for (y = 0; y < 136; y += 2) { //every 2 pixel in height
        c = pix(x, y) //take color
        c = (c + 1) % 15 //change it
        pix(x, y, c) //put it back
      }
    }
  }
  t++
}
```

line

рисование линии

line (`x0, y0, x1, y1, color` **)**

Параметры:

`x0` - координата начала линии по оси x

`y0` - координата начала линии по оси y

`x1` - координата конца линии по оси x

`y1` - координата конца линии по оси y

`color` - индекс цвета в текущей палитре

Описание:

Рисует прямую цветную линию начиная с координат `(x0,y0)` до `(x1,y1)`.

Пример:

Запустить или скачать картридж примера.

```
-- title:  line demo
-- author:
-- desc:
-- script: lua
-- input:  gamepad

pi8=math.pi/8
pi2=math.pi*2

t=0

function TIC()
 cls()

 --lines
 for i=t%8,135,8 do
  line(i,0,0,135-i,8)
```

```
  line(i,135,135,135-i,6)
  t=t+0.01
 end

 --prism
 for i=t/16%pi8,pi2,pi8 do
  x=68+32*math.cos(i)
  y=68+32*math.cos(i)
  line(135,0,x,y,15)
  line(0,135,x,y,15)
 end

 --Border
 line(0,0,135,0,8)
 line(0,0,0,135,8)
 line(135,0,135,135,6)
 line(0,135,135,135,6)

end
```

Запустить или скачать картридж примера.

```
-- title:  line demo
-- author:
-- desc:
-- script: moon
-- input:  gamepad

pi8=math.pi/8
pi2=math.pi*2

t=0
export TIC=->
  cls()

  --lines
  for i=t%8,135,8
    line(i,0,0,135-i,8)
    line(i,135,135,135-i,6)
    t+=0.01

  --prism
  for i=t/16%pi8,pi2,pi8
    x=68+32*math.cos(i)
    y=68+32*math.cos(i)
    line 135,0,x,y,15
    line 0,135,x,y,15

  --Border
  line 0,0,135,0,8
  line 0,0,0,135,8
  line 135,0,135,135,6
  line 0,135,135,135,6
```

```
// title: line demo
// author:
// desc:
// script: js
// input: gamepad

pi8 = Math.PI / 8
pi2 = Math.PI * 2
```

```
t = 0

function TIC() {
  cls()

  //lines
  for (i = t % 8; i < 135; i += 8) {
    line(i, 0, 0, 135 - i, 8)
    line(i, 135, 135, 135 - i, 6)
    t = t + 0.01
  }

  //prism
  for (i = t / 16 % pi8; i < pi2; i += pi8) {
    x = 68 + 32 * Math.cos(i)
    y = 68 + 32 * Math.cos(i)
    line(135, 0, x, y, 15)
    line(0, 135, x, y, 15)
  }

  //Border
  line(0, 0, 135, 0, 8)
  line(0, 0, 0, 135, 8)
  line(135, 0, 135, 135, 6)
  line(0, 135, 135, 135, 6)

}
```

circ

рисование круга

circ (`x, y, radius, color` **)**

Параметры:

`x` - координата центра круга по оси x

`y` - координата центра круга по оси y

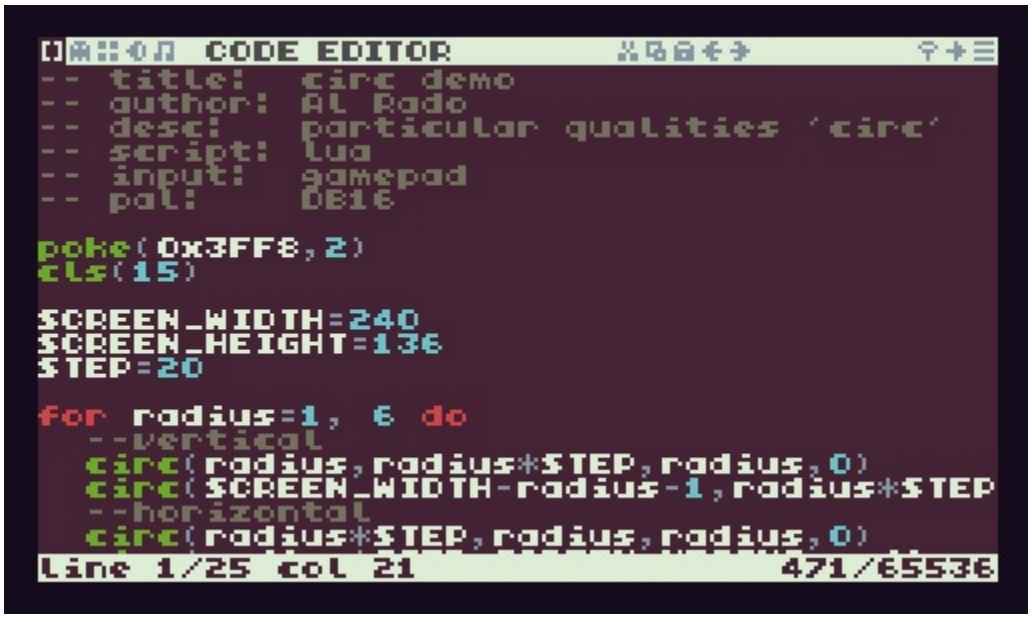
`radius` - радиус круга в пикселях

`color` - индекс цвета в текущей палитре

Описание:

Рисует заполненный цветной круг с центром `x` и `y` с указанным радиусом. Используется алгоритм "bresenham".

Пример:



Запустить или скачать картридж примера.

```
-- title:  circ demo
-- author: Al Rado
-- desc:   particular qualities 'circ'
-- script: lua
-- input:  gamepad

poke(0x3FF8,2)
cls(15)

SCREEN_WIDTH=240
SCREEN_HEIGHT=136
STEP=20

for radius=1, 6 do
  --vertical
  circ(radius,radius*STEP,radius,0)
  circ(SCREEN_WIDTH-radius-1,radius*STEP,radius,0)-- minus one!
```

```
  --horizontal
  circ(radius*STEP,radius,radius,0)
  circ(radius*STEP,SCREEN_HEIGHT-radius-1,radius,0)-- minus one!
end

function TIC() end
```

Пример:

Запустить или скачать картридж примера.

```
-- title:  circ demo
-- author: Filippo
-- desc:
-- script: lua
-- input:  gamepad

--init balls
balls={}
d=1
for i=0,50 do
  ball={
    x =math.random(10,220),
    y =math.random(10,126),
    dx=math.random(1,2)*d,
    dy=math.random(1,2)*d,
    r =math.random(6,12),
    c =math.random(1,6)
  }
  balls[i]=ball
  d=d*-1
end

function TIC()
  cls()
  for k,b in pairs(balls) do
    --move the ball
    b.x=b.x+b.dx
    b.y=b.y+b.dy
    --check right/left walls
    if b.x >= 240-b.r then
      b.x=240-b.r-1 --constraints inside the wall
      b.dx=-b.dx  --reverse direction
```

```
    elseif b.x < b.r then
      b.x=b.r
      b.dx=-b.dx
    end
    --check bottom/top walls
    if b.y >= 136-b.r then
      b.y=136-b.r-1
      b.dy=-b.dy
    elseif b.y < b.r then
      b.y=b.r
      b.dy=-b.dy
    end
    --draw balls
    circ(b.x,b.y,b.r,b.c)
    circ(b.x+b.r/4,b.y-b.r/4,b.r/4,b.c+7)
 end
end
```

Запустить или скачать картридж примера.

```
-- title:  circ demo
-- author: Al Rado
-- desc:   particular qualities 'circ'
-- script: moon
-- input:  gamepad

poke 0x3FF8,2
cls 15

SCREEN_WIDTH=240
SCREEN_HEIGHT=136
STEP=20

for radius=1, 6
  --vertical
  circ(radius,radius*STEP,radius,0)
  circ(SCREEN_WIDTH-radius-1,radius*STEP,radius,0)-- minus one!
  --horizontal
  circ(radius*STEP,radius,radius,0)
  circ(radius*STEP,SCREEN_HEIGHT-radius-1,radius,0)-- minus one!

export TIC=->
```

Пример:

Запустить или скачать картридж примера.

```
-- title:  circ demo
-- author: Filippo
-- desc:
-- script: moon
-- input:  gamepad

--init balls
balls={}
d=1
for i=0,50
  ball={
    x:math.random(10,220)
    y:math.random(10,126)
    dx:math.random(1,2)*d
    dy:math.random(1,2)*d
    r:math.random(6,12)
    c:math.random(1,6)
  }
  balls[i]=ball
  d*=-1

export TIC=->
  cls()
  for b in *balls
    --move the ball
    b.x=b.x+b.dx
    b.y=b.y+b.dy
    --check right/left walls
    if b.x >= 240-b.r
      b.x=240-b.r-1 --constraints inside the wall
      b.dx=-b.dx  --reverse direction
    elseif b.x < b.r
      b.x=b.r
      b.dx=-b.dx
    --check bottom/top walls
    if b.y >= 136-b.r
      b.y=136-b.r-1
      b.dy=-b.dy
    elseif b.y < b.r
```

```
      b.y=b.r
      b.dy=-b.dy
    --draw balls
    circ b.x,b.y,b.r,b.c
    circ b.x+b.r/4,b.y-b.r/4,b.r/4,b.c+7
```

```
// title:  circ demo
// author: Al Rado
// desc:   particular qualities 'circ'
// script: js
// input:  gamepad

poke(0x3FF8, 2)
cls(15)

SCREEN_WIDTH = 240
SCREEN_HEIGHT = 136
STEP = 20

for (radius = 1; radius < 6; radius++) {
  //vertical
  circ(radius, radius * STEP, radius, 0)
  circ(SCREEN_WIDTH - radius - 1, radius * STEP, radius, 0)// minus one!
  //horizontal
  circ(radius * STEP, radius, radius, 0)
  circ(radius * STEP, SCREEN_HEIGHT - radius - 1, radius, 0)// minus one!
}

function TIC() { }
```

Пример:


```
// title:  circ demo
// author: Filippo
// desc:
// script: js
// input:  gamepad

//init balls
balls = {}
d = 1
```

```
for (i = 0; i < 50; i++) {
  ball = {
    x: random(10, 220),
    y: random(10, 126),
    dx: random(1, 2) * d,
    dy: random(1, 2) * d,
    r: random(6, 12),
    c: random(1, 6)
  }
  balls[i] = ball
  d = d * -1
}

function random(min, max) {
  return Math.min(Math.random() * max + min, max)
}

function TIC() {
  cls()
  for (var k in balls) {
    b = balls[k]
    //move the ball
    b.x = b.x + b.dx
    b.y = b.y + b.dy
    //check right/left walls
    if (b.x >= 240 - b.r) {
      b.x = 240 - b.r - 1 //constraints inside the wall
      b.dx = -b.dx  //reverse direction
    }
    else if (b.x < b.r) {
      b.x = b.r
      b.dx = -b.dx
    }
    //check bottom/top walls
    if (b.y >= 136 - b.r) {
      b.y = 136 - b.r - 1
      b.dy = -b.dy
    }
    else if (b.y < b.r) {
      b.y = b.r
      b.dy = -b.dy
    }
    //draw balls
    circ(b.x, b.y, b.r, b.c)
    circ(b.x + b.r / 4, b.y - b.r / 4, b.r / 4, b.c + 7)
  }
}
```

circb

рисование окружности

circb (`x, y, radius, color` **)**

Параметры:

`x` - координата центра круга по оси x

`y` - координата центра круга по оси y

`radius` - радиус круга в пикселях

`color` - индекс цвета в текущей палитре

Описание:

Рисует цветную окружность с центром `x` и `y` с указанным радиусом. Используется алгоритм "bresenham".

Пример:

Запустить или скачать картридж примера.

```
-- title:  circb demo
-- author: Filippo
-- desc:
-- script: lua
-- input:  gamepad

a=0
space=10

function TIC()
  cls()
  for i=0,200,space do
    circb(120+80*math.sin(a),
          68+40*math.cos(a),
          i+time()/40%space,
          8)
    circb(120+80*math.sin(a/2),
```

```
          68+40*math.cos(a/2),
          i+time()/40%space,
          8)
  end
  a=a+math.pi/240
end
```

Запустить или скачать картридж примера.

```
-- title:  circb demo
-- author: Filippo
-- desc:
-- script: moon
-- input:  gamepad

a=0
space=10

export TIC=->
  cls()
  for i=0,200,space
    circb(120+80*math.sin(a),
          68+40*math.cos(a),
          i+time()/40%space,
          8)
    circb(120+80*math.sin(a/2),
          68+40*math.cos(a/2),
          i+time()/40%space,
          8)
  a=a+math.pi/240
```

```
// title:  circb demo
// author: Filippo
// desc:
// script: js
// input:  gamepad

a = 0.0
space = 10

function TIC() {
  cls()
  for (i = 0; i < 200; i += space) {
    circb(120 + 80 * Math.sin(a),
      68 + 40 * Math.cos(a),
      i + time() / 40 % space,
      8)
    circb(120 + 80 * Math.sin(a / 2),
      68 + 40 * Math.cos(a / 2),
      i + time() / 40 % space,
      8)
  }
  a += Math.PI / 240
}
```

rect

рисование заполненного прямоугольника

rect (`x, y, w, h, color` **)**

Параметры:

`x` - координата левого-верхнего угла прямоугольника по оси x

`y` - координата левого-верхнего угла прямоугольника по оси y

`w` - ширина прямоугольника в пикселях

`h` - высота прямоугольника в пикселях

`color` - индекс цвета в текущей палитре, который будет использован при заливке прямоугольника

Описание:

Эта функция рисует цветной заполненный прямоугольник по указанным координатам.

Если Вам нужно отрисовать только рамку, используйте функцию rectb

Пример:

Запустить или скачать картридж примера.

```
-- title:  rect demo
-- author: Filippo
-- desc:
-- script: lua
-- input:  gamepad

x=120
y=68
dx=7
dy=4
col=1

cls()

function TIC()
  --Update x/y
```

```
  x=x+dx
  y=y+dy
  --Check screen walls
  if x>240-6 or x<0 then
    dx=-dx
    col=col%15+1
  end
  if y>136-6 or y<0 then
    dy=-dy
    col=col%15+1
  end
  --Draw rectangle
  rect (x,y,6,6,col)
end
```

Запустить или скачать картридж примера.

```
-- title:  rect demo
-- author: Filippo
-- desc:
-- script: moon
-- input:  gamepad

x=120
y=68
dx=7
dy=4
col=1

cls()

export TIC=->
  --Update x/y
  x+=dx
  y+=dy
  --Check screen walls
  if x>240-6 or x<0
    dx=-dx
    col=col%15+1
  if y>136-6 or y<0
    dy=-dy
    col=col%15+1
  --Draw rectangle
  rect x,y,6,6,col
```

```
// title: rect demo
// author: Filippo
// desc:
// script: js
// input: gamepad

x = 120
y = 68
dx = 7
dy = 4
col = 1

cls()

function TIC() {
  //Update x/y
  x += dx
  y += dy
```

```
  //Check screen walls
  if (x > 240 - 6 || x < 0) {
    dx = -dx
    col = col % 15 + 1
  }
  if (y > 136 - 6 || y < 0) {
    dy = -dy
    col = col % 15 + 1
  }
  //Draw rectangle
  rect(x, y, 6, 6, col)
}
```

rectb

рисование прямоугольника

rectb (`x, y, w, h, color` **)**

Параметры:

`x` - координата левого-верхнего угла прямоугольника по оси x

`y` - координата левого-верхнего угла прямоугольника по оси y

`w` - ширина прямоугольника в пикселях

`h` - высота прямоугольника в пикселях

`color` - индекс цвета в текущей палитре, который будет использован при заливке прямоугольника

Описание:

Эта функция рисует цветную рамку-прямоугольник по указанным координатам.

Если Вам нужно отрисовать заполненный цветом прямоугольник, используйте функцию rect

Пример:

Запустить или скачать картридж примера.

```
-- title:  rectb demo
-- author: Filippo
-- desc:
-- script: lua
-- input:  gamepad

x=104
y=60

function TIC()
  cls()
  for s=280,0,-4 do
    s2=s/2
    sd=500/s
    x=sd*math.sin(time()/1000)
    y=sd*math.cos(time()/1000)
```

```
    rectb(120+x-s2,68+y-(s2/2),s,s2,8)
  end
end
```

Запустить или скачать картридж примера.

```
-- title:  rectb demo
-- author: Filippo
-- desc:   code refactored by Al Rado
-- script: lua
-- input:  gamepad

HALF_SCR_W = 240/2
HALF_SCR_H = 136/2
DEVIATION = 150
SPEED = 1/500
RECT_COUNT = 70
RECT_STEP = 4
RECT_COLOR = 8

function TIC()
  cls()
  for i = 1, RECT_COUNT do
    width = i*RECT_STEP
    height = width/2
    slowedTime = time()*SPEED
    x = math.sin(slowedTime) * DEVIATION/i - width/2
    y = math.cos(slowedTime) * DEVIATION/i - height/2
    rectb(HALF_SCR_W+x, HALF_SCR_H+y, width, height, RECT_COLOR)
  end
end
```

Запустить или скачать картридж примера.

```
-- title:  rectb demo
-- author: Filippo
-- desc:
-- script: moon
-- input:  gamepad

x=104
y=60

export TIC=->
  cls()
  for s=280,0,-4
    s2=s/2
    sd=500/s
    x=sd*math.sin(time()/1000)
    y=sd*math.cos(time()/1000)
    rectb 120+x-s2,68+y-(s2/2),s,s2,8
```

Запустить или скачать картридж примера.

```
-- title:  rectb demo
-- author: Filippo
-- desc:   code refactored by Al Rado
-- script: moon
-- input:  gamepad

HALF_SCR_W = 240/2
```

```
HALF_SCR_H = 136/2
DEVIATION = 150
SPEED = 1/500
RECT_COUNT = 70
RECT_STEP = 4
RECT_COLOR = 8

export TIC=->
  cls()
  for i = 1, RECT_COUNT
    width = i*RECT_STEP
    height = width/2
    slowedtime = time()*SPEED
    x = math.sin(slowedtime) * DEVIATION/i - width/2
    y = math.cos(slowedtime) * DEVIATION/i - height/2
    rectb HALF_SCR_W+x, HALF_SCR_H+y, width, height, RECT_COLOR
```

```
// title: rectb demo
// author: Filippo
// desc:
// script: js
// input: gamepad

x = 104
y = 60

function TIC() {
  cls()
  for (s = 280; s > 0; s -= 4) {
    s2 = s / 2
    sd = 500 / s
    x = sd * Math.sin(time() / 1000)
    y = sd * Math.cos(time() / 1000)
    rectb(120 + x - s2, 68 + y - (s2 / 2), s, s2, 8)
  }
}
```

```
// title: rectb demo
// author: Filippo
// desc: code refactored by Al Rado
// script: js
// input: gamepad

HALF_SCR_W = 240 / 2
HALF_SCR_H = 136 / 2
DEVIATION = 150
SPEED = 1 / 500
RECT_COUNT = 70
RECT_STEP = 4
RECT_COLOR = 8

function TIC() {
  cls()
  for (i = 1; i < RECT_COUNT; i++) {
    width = i * RECT_STEP
    height = width / 2
    slowedTime = time() * SPEED
    x = Math.sin(slowedTime) * DEVIATION / i - width / 2
    y = Math.cos(slowedTime) * DEVIATION / i - height / 2
    rectb(HALF_SCR_W + x, HALF_SCR_H + y, width, height, RECT_COLOR)
  }
}
```

tri

рисование заполненного треугольника

tri (`x1, y1, x2, y2, x3, y3, color` **)**

Параметры:

`x1` - координата по оси x первого угла треугольника

`y1` - координата по оси y первого угла треугольника

`x2` - координата по оси x второго угла треугольника

`y2` - координата по оси y второго угла треугольника

`x3` - координата по оси x третьего угла треугольника

`y3` - координата по оси y третьего угла треугольника

`color` - индекс цвета в текущей палитре

Описание:

Рисует треугольник заполненный цветом.

Пример:

Запустить или скачать картридж примера.

```
-- title:  tri demo
-- author: Filippo
-- desc:
-- script: lua
-- input:  gamepad

function Pir(x,y,w,h,cx,cy)
  tri(x,y,w/2+cx,h/2+cy,x+w,y,1)
  tri(x+w,y,w/2+cx,h/2+cy,x+w,y+h,2)
  tri(x,y,w/2+cx,h/2+cy,x,y+h,8)
  tri(x,y+h,w/2+cx,h/2+cy,x+w,y+h,15)
end

cls()
```

```
function TIC()
  for x=0,240,28 do
    for y=0,136,28 do
      cx=12*math.sin(time()/30000*(x+y+1))
      cy=12*math.cos(time()/30000*(x+y+1))
      Pir(x,y,25,25,x+cx,y+cy)
    end
  end
end
```

Запустить или скачать картридж примера.

```
-- title:  tri demo
-- author: Filippo
-- desc:
-- script: moon
-- input:  gamepad

Pir=(x,y,w,h,cx,cy)->
  tri x,y,w/2+cx,h/2+cy,x+w,y,1
  tri x+w,y,w/2+cx,h/2+cy,x+w,y+h,2
  tri x,y,w/2+cx,h/2+cy,x,y+h,8
  tri x,y+h,w/2+cx,h/2+cy,x+w,y+h,15

cls()

export TIC=->
  for x=0,240,28 do
    for y=0,136,28 do
      cx=12*math.sin(time()/30000*(x+y+1))
      cy=12*math.cos(time()/30000*(x+y+1))
      Pir(x,y,25,25,x+cx,y+cy)
```

```
// title: tri demo
// author: Filippo
// desc:
// script: js
// input: gamepad

function Pir(x, y, w, h, cx, cy) {
  tri(x, y, w / 2 + cx, h / 2 + cy, x + w, y, 1)
  tri(x + w, y, w / 2 + cx, h / 2 + cy, x + w, y + h, 2)
  tri(x, y, w / 2 + cx, h / 2 + cy, x, y + h, 8)
  tri(x, y + h, w / 2 + cx, h / 2 + cy, x + w, y + h, 15)
}

cls()

function TIC() {
  for (x = 0; x < 240; x += 28) {
    for (y = 0; y < 136; y += 28) {
      cx = 12 * Math.sin(time() / 30000 * (x + y + 1))
      cy = 12 * Math.cos(time() / 30000 * (x + y + 1))
      Pir(x, y, 25, 25, x + cx, y + cy)
    }
  }
}
```

Экран

Функции для работы с экраном.

cls

очистка экрана

cls (`[color]` **)**

Параметры:

`color` - индекс цвета в загруженной палитре. Начинается с нуля.

Описание:

При вызове этой функции очищается весь экран и заливается цветом указанном в качестве параметра. По умолчанию, используется первый цвет (индекс = 0). Как правило, вызывается из функции `TIC`, но это не является обязательным условием. Вы можете сделать какие-нибудь странные эффекты либо мерцающий экран используя её.

Подсказка:

Используйте индексы цветов свыше 15, чтобы получить специальные заполняющие паттерны.

Пример:

Запустить или скачать картридж примера.

```
-- title:  cls demo
-- author:
-- desc:
-- script: lua
-- input:  gamepad

c=0

function TIC()
  --use up/down to change color
  if btnp(0) then c=c+1 end
  if btnp(1) then c=c-1 end

  --clear with the color
  cls(c)
```

```
  --make a background for the text
  rect(0,0,240,8,0)
  --ouput a text with function call
  print('cls('..c..')  --use up/down to change color')
end
```

Запустить или скачать картридж примера.

```
-- title:  cls demo
-- author:
-- desc:
-- script: moon
-- input:  gamepad

c=0

export TIC=->
  --Use Up/Down to change color
  if btnp(0) then c=c+1
  if btnp(1) then c=c-1

  --Clear with the color
  cls c

  --Make a background for the text
  rect 0,0,240,8,0
  --Ouput a text with function call
  print 'cls('..c..')  --Use Up/Down to change color'
```

```
// title: cls demo
// author:
// desc:
// script: js
// input: gamepad

c = 0

function TIC() {
  //use up/down to change color
  if (btnp(0)) c++
  if (btnp(1)) c--

  //clear with the color
  cls(c)

  //make a background for the text
  rect(0, 0, 240, 8, 0)
  //ouput a text with function call
  print('cls(' + c + ') --use up/down to change color')
}
```

clip

ограничение вывода на экран

clip (`[x, y, width, height]` **)**

Параметры:

`x` - координата по оси x, начало координат левый-верхний угол экрана

`y` - координата по оси y, начало координат левый-верхний угол экрана

`width` - ширина области

`height` - высота области

Описание:

Эта функция ограничивает то, что рисуется на экране параметрами ограничивающего прямоугольника. Всё что выходит за границы, не будет отображаться на экране.

Память

Функции для чтения, записи и копирования значений в памяти.

peek

чтение байта из памяти

peek (`addr` **) ->** `val`

Параметры:

`addr` - любой адрес из пространства 64Kb памяти, из которого Вам нужно прочесть значение байта

Возвращает:

`val` - значение байта по указанному адресу

Описание:

Эта функция позволяет читать значения памяти TIC - байт.

Она удобна для доступа к ресурсам, созданным с помощью интегрированных средств, таких как спрайты, карты, звуки, данные картриджа.

Адрес указывается в шестнадцатеричном формате, но возвращаемое значение в десятичном.

Пример:

Запустить или скачать картридж примера.

```
-- title:  peek/poke demo
-- author:
-- desc:
-- script: lua
-- input:  gamepad

-- use poke to create a sprite from a hexadecimal data string
img="a000000aa0f00f0aaaaaaaaaaafaaa6aafffaaaaaafaa6aaaaaaaaaa33333333"
-- foreach of the 32 bytes in the sprite
for i=0,31 do
  --extract the data from the string 2 chars at time
  s = string.sub(img,i*2+1,i*2+2)
  --convert the hex string to a decimal number
  val = tonumber(s,16)
  --put the value in the RAM area dedicated to sprites, sprite number 4
```

```
  poke(0x4000+128+i,val)
end
sync()
-- read back the sprite data and print it to console
msg=''
-- foreach of the 32 bytes in the sprite
for i=0,31 do
  --read the byte value in decimal format
  val=peek(0x4000+128+i)
  -- convert the value in hexadecimal format
  s=string.format("%02x",val)
  -- concatenate the string to a final message
  msg=msg..s
end

-- print it to the console
trace(msg)

function TIC()
  cls()
  print('Done.. have a look at the sprite editor.',0,10)
  print('And to the console also.',0,20)
end
```

Запустить или скачать картридж примера.

```
-- title:  peek/poke demo
-- author:
-- desc:
-- script: moon
-- input:  gamepad

-- use poke to create a sprite from a hexadecimal data string
img="a000000aa0f00f0aaaaaaaaaaafaaa6aafffaaaaaafaa6aaaaaaaaa33333333"
-- foreach of the 32 bytes in the sprite
for i=0,31
  --extract the data from the string 2 chars at time
  s = string.sub(img,i*2+1,i*2+2)
  --convert the hex string to a decimal number
  val = tonumber(s,16)
  --put the value in the RAM area dedicated to sprites, sprite number 4
  poke(0x4000+128+i,val)
sync()
-- read back the sprite data and print it to console
msg=''
-- foreach of the 32 bytes in the sprite
for i=0,31
  --read the byte value in decimal format
  val=peek(0x4000+128+i)
  -- convert the value in hexadecimal format
  s=string.format("%02x",val)
  -- concatenate the string to a final message
  msg..=s

-- print it to the console
trace msg

export TIC=->
  cls()
  print 'Done.. have a look at the sprite editor.',0,10
  print 'And to the console also.',0,20
```

```
// title: peek/poke demo
```

```
// author:
// desc:
// script: js
// input: gamepad

// use poke to create a sprite from a hexadecimal data string
img = "a000000aa0f00f0aaaaaaaaaaafaaa6aafffaaaaaafaa6aaaaaaaaaa33333333"
// foreach of the 32 bytes in the sprite
for (i = 0; i < 32; i++) {
  //extract the data from the string 2 chars at time
  s = img[i * 2 + 1] + img[i * 2]
  //convert the hex string to a decimal number
  val = parseInt(s, 16)
  //put the value in the RAM area dedicated to sprites, sprite number 4
  poke(0x4000 + 128 + i, val)
}
sync()

// read back the sprite data and print it to console
msg = ''
// foreach of the 32 bytes in the sprite
for (i = 0; i < 32; i++) {
  //read the byte value in decimal format
  val = peek(0x4000 + 128 + i)
  // convert the value in hexadecimal format
  s = val.toString(16)
  // concatenate the string to a final message
  msg += s
}

// print it to the console
trace(msg)

function TIC() {
  cls()
  print('Done+ have a look at the sprite editor.', 0, 10)
  print('And to the console also.', 0, 20)
}
```

poke

запись байта в память

poke (`addr, val` **)**

Параметры:

`addr` - любой адрес из пространства 64Kb памяти, в который Вам нужно записать значение байта

`val` - значение которое нужно записать

Описание:

Эта функция позволяет записывать значения в память TIC - байт.

Адрес указывается в шестнадцатеричном формате, но возвращаемое значение в десятичном.

Пример:

Запустить или скачать картридж примера.

```
-- title:  poke demo
-- author: Al Rado
-- desc:   Set sprite from text to sprite sheet
-- script: lua
-- input:  gamepad

-- количество полубайт(nibble) в спрайте
NIBBLES_IN_SPR=8*8
-- количество байт в спрайте
BYTES_IN_SPR=NIBBLES_IN_SPR/2
-- адрес спрайт-листа в памяти
SPR_SHEET_ADDR=0x4000
-- индекс спрайта в спрайт-листе
sprIx=5
-- данные спрайта в шестнадцатеричном виде, каждый символ соответствует полубайту - индексу цвета(0-f)
-- можно получить, просто скопировав в буфер обмена в редакторе спрайтов
sprDataHex="a000000aa0f00f0aaaaaaaaaaafaaa6aafffaaaaaafaa6aaaaaaaaaa33333333"

cls()
```

```
for i=1,BYTES_IN_SPR do
  byteStr=sprDataHex:sub(i*2-1,i*2):reverse()
  byte=tonumber(byteStr,16)
  poke(SPR_SHEET_ADDR+(BYTES_IN_SPR*sprIx)+i-1,byte)
end

print("See sprite sheet, index = "..sprIx)

function TIC() end
```

Пример 2:

Запустить или скачать картридж примера.

```
-- title:  poke demo
-- author: Filippo
-- desc:
-- script: lua
-- input:  gamepad

l1=49
w1=18
l2=21
w2=12

function TIC()

  --Video noise
  for i=0,(240*136)/2-1 do
    poke(0x0000+i,(i*i*time())/3000000000%2+1)
  end

  --Sounds
  if btnp(0,3,3)then l1=l1-1 end
  if btnp(1,3,3)then l1=l1+1 end
  if btnp(2,3,3)then w1=w1-1 end
  if btnp(3,3,3)then w1=w1+1 end

  l1=l1<136 and l1 or 135
  l1=l1>0 and l1 or 0
  w1=w1<120 and w1 or 112
  w1=w1>2 and w1 or 2
```

```
  if btnp(8,3,3)then l2=l2-1 end
  if btnp(9,3,3)then l2=l2+1 end
  if btnp(10,3,3)then w2=w2-1 end
  if btnp(11,3,3)then w2=w2+1 end

  l2=l2<136 and l2 or 135
  l2=l2>0 and l2 or 0
  w2=w2<120 and w2 or 112
  w2=w2>2 and w2 or 2

  --S1
  off1=l1*120
  m=0
  for i=60-w1/2,60+w1/2 do
    m=m+peek(0x0000+off1+i)
  end

  f=(10+((1+m)/w1))*800
  f=math.floor(f)
  print(l1..'-'..w1..'-'..f,0,0)
  poke(0xFF80+0,f&0XFF)
  poke(0xFF80+2,(f&0XFF00)>>8)

  --Draw line
  for i=60-w1/2,60+w1/2 do
    poke(0x0000+off1+i,136)
  end

  --S2
  off2=l2*120
  m=0
  for i=60-w2/2,60+w2/2 do
    m=m+peek(0x0000+off2+i)
  end

  f=(10+((1+m)/w2))*800
  f=math.floor(f)
  print(l2..'-'..w2..'-'..f,0,10)
  poke(0xFF80+8,f&0XFF)
  poke(0xFF80+10,(f&0XFF00)>>8)

  --Draw line
  for i=60-w2/2,60+w2/2 do
    poke(0x0000+off2+i,102)
  end

end
```

Запустить или скачать картридж примера.

```
-- title:  poke demo
-- author: Al Rado
-- desc:   Set sprite from text to sprite sheet
-- script: moon
-- input:  gamepad

-- количество полубайт(nibble) в спрайте
NIBBLES_IN_SPR=8*8
-- количество байт в спрайте
BYTES_IN_SPR=NIBBLES_IN_SPR/2
-- адрес спрайт-листа в памяти
SPR_SHEET_ADDR=0x4000
-- индекс спрайта в спрайт-листе
sprIx=5
-- данные спрайта в шестнадцатеричном виде, каждый символ соответствует полубайту - индексу цвета(0-f)
-- можно получить, просто скопировав в буфер обмена в редакторе спрайтов
sprDataHex="a000000aa0f00f0aaaaaaaaaaafaaa6aafffaaaaaafaa6aaaaaaaaa33333333"

cls()
for i=1,BYTES_IN_SPR
  byteStr=sprDataHex\sub(i*2-1,i*2)\reverse()
  byte=tonumber(byteStr,16)
  poke(SPR_SHEET_ADDR+(BYTES_IN_SPR*sprIx)+i-1,byte)

print("See sprite sheet, index = "..sprIx)

export TIC=->
```

Пример 2:

Запустить или скачать картридж примера.

```
-- title:  poke demo
-- author: Filippo
-- desc:
-- script: moon
-- input:  gamepad

l1=49
w1=18
l2=21
w2=12

export TIC=->

  --Video noise
  for i=0,(240*136)/2-1
    poke(0x0000+i,(i*i*time())/3000000000%2+1)

  --Sounds
  if btnp(0,3,3) then l1=l1-1
  if btnp(1,3,3) then l1=l1+1
  if btnp(2,3,3) then w1=w1-1
  if btnp(3,3,3) then w1=w1+1

  l1=l1<136 and l1 or 135
  l1=l1>0 and l1 or 0
  w1=w1<120 and w1 or 112
  w1=w1>2 and w1 or 2

  if btnp(8,3,3)then l2=l2-1
  if btnp(9,3,3)then l2=l2+1
  if btnp(10,3,3)then w2=w2-1
  if btnp(11,3,3)then w2=w2+1

  l2=l2<136 and l2 or 135
  l2=l2>0 and l2 or 0
  w2=w2<120 and w2 or 112
  w2=w2>2 and w2 or 2

  --S1
  off1=l1*120
  m=0
```

```
  for i=60-w1/2,60+w1/2
    m=m+peek(0x0000+off1+i)

  f=(10+((1+m)/w1))*800
  f=math.floor(f)
  print l1..'-'..w1..'-'..f,0,0
  poke 0xFF80+0,f&0xFF
  poke 0xFF80+2,(f&0xFF00)>>8

  --Draw line
  for i=60-w1/2,60+w1/2
    poke 0x0000+off1+i,136

  --S2
  off2=l2*120
  m=0
  for i=60-w2/2,60+w2/2
    m=m+peek 0x0000+off2+i

  f=(10+((1+m)/w2))*800
  f=math.floor(f)
  print l2..'-'..w2..'-'..f,0,10
  poke 0xFF80+8,f&0xFF
  poke 0xFF80+10,(f&0xFF00)>>8

  --Draw line
  for i=60-w2/2,60+w2/2
    poke 0x0000+off2+i,102
```

```
// title: poke demo
// author: Al Rado
// desc: Set sprite from text to sprite sheet
// script: js
// input: gamepad

// количество полубайт(nibble) в спрайте
NIBBLES_IN_SPR = 8 * 8
// количество байт в спрайте
BYTES_IN_SPR = NIBBLES_IN_SPR / 2
// адрес спрайт-листа в памяти
SPR_SHEET_ADDR = 0x4000
// индекс спрайта в спрайт-листе
sprIx = 5
// данные спрайта в шестнадцатеричном виде, каждый символ соответствует полубайту - индексу цвета(0-f)
// можно получить, просто скопировав в буфер обмена в редакторе спрайтов
sprDataHex = "a000000aa0f00f0aaaaaaaaaaafaaa6aafffaaaaaafaa6aaaaaaaaaa33333333"

cls()
for (i = 0; i < BYTES_IN_SPR; i++) {
  byteStr = "" + sprDataHex[i * 2 + 1] + sprDataHex[i * 2]
  byte = parseInt(byteStr, 16)
  poke(SPR_SHEET_ADDR + (BYTES_IN_SPR * sprIx) + i, byte)
}
sync()

print("See sprite sheet, index = " + sprIx)

function TIC() { }
```

Пример 2:

```
// title: poke demo
// author: Filippo
```

```
// desc:
// script: js
// input: gamepad

l1 = 49
w1 = 18
l2 = 21
w2 = 12

function TIC() {

  //Video noise
  for (i = 0; i < (240 * 136) / 2 - 1; i++) {
    poke(0x0000 + i, (i * i * time()) / 3000000000 % 2 + 1)
  }

  //Sounds
  if (btnp(0, 3, 3)) { l1 = l1 - 1 }
  if (btnp(1, 3, 3)) { l1 = l1 + 1 }
  if (btnp(2, 3, 3)) { w1 = w1 - 1 }
  if (btnp(3, 3, 3)) { w1 = w1 + 1 }

  l1 = l1 < 136 ? l1 : 135
  l1 = l1 > 0 ? l1 : 0
  w1 = w1 < 120 ? w1 : 112
  w1 = w1 > 2 ? w1 : 2

  if (btnp(8, 3, 3)) { l2 = l2 - 1 }
  if (btnp(9, 3, 3)) { l2 = l2 + 1 }
  if (btnp(10, 3, 3)) { w2 = w2 - 1 }
  if (btnp(11, 3, 3)) { w2 = w2 + 1 }

  l2 = l2 < 136 ? l2 : 135
  l2 = l2 > 0 ? l2 : 0
  w2 = w2 < 120 ? w2 : 112
  w2 = w2 > 2 ? w2 : 2

  //S1
  off1 = l1 * 120
  m = 0
  for (i = 60 - w1 / 2; i < 60 + w1 / 2; i++) {
    m = m + peek(0x0000 + off1 + i)
  }

  f = (10 + ((1 + m) / w1)) * 800
  f = Math.floor(f)
  print(l1 + '-' + w1 + '-' + f, 0, 0)
  poke(0xFF80 + 0, f & 0XFF)
  poke(0xFF80 + 2, (f & 0XFF00) >> 8)

  //Draw line
  for (i = 60 - w1 / 2; i < 60 + w1 / 2; i++) {
    poke(0x0000 + off1 + i, 136)
  }

  //S2
  off2 = l2 * 120
  m = 0
  for (i = 60 - w2 / 2; i < 60 + w2 / 2; i++) {
    m = m + peek(0x0000 + off2 + i)
  }

  f = (10 + ((1 + m) / w2)) * 800
  f = Math.floor(f)
  print(l2 + '-' + w2 + '-' + f, 0, 10)
  poke(0xFF80 + 8, f & 0XFF)
```

```
  poke(0xFF80 + 10, (f & 0XFF00) >> 8)

  //Draw line
  for (i = 60 - w2 / 2; i < 60 + w2 / 2; i++) {
    poke(0x0000 + off2 + i, 102)
  }

}
```

peek4

чтение полубайта из памяти

peek4 (`addr` **) ->** `val`

Параметры:

`addr` - любой адрес из пространства 64Kb памяти, из которого Вам нужно прочесть значение полубайта(4 бита)

Возвращает:

`val` - значение полубайта(4 бита) по указанному адресу

Описание:

Эта функция позволяет читать значения памяти TIC - полубайт.
Она используется для доступа к ресурсам, созданным с помощью интегрированных средств, таких например как спрайты в спрайт-листе.
Адрес указывается в шестнадцатеричном формате, но возвращаемое значение в десятичном.
Стоит также отметить, что peek4 и poke4 оперируют полубайтами (4 бита), поэтому адрес умножается на два по отношению к обычным peek и poke, которые оперируют байтами (8 бит).

```
+----------+----------+----------+----------+
|   4bit   |   4bit   |   4bit   |   4bit   |
+----------+----------+----------+----------|
|         8bit        |         8bit        |
+---------------------+---------------------+
```

Пример:

```lua
-- script: lua
-- возвращает значение пикселя в спрайт-листе

function sget(x,y)
    local addr=0x4000+(x//8+y//8*16)*32 -- получить адрес спрайта
    return peek4(addr*2+x%8+y%8*8) -- получить значение пикселя в спрайт листе
end

-- тут Ваш код
```

```javascript
// script: js
// возвращает значение пикселя в спрайт-листе

function sget(x,y)
    local addr=0x4000+(x/8+y/8*16)*32 // получить адрес спрайта
    return peek4(addr*2+x%8+y%8*8) // получить значение пикселя в спрайт листе
end

-- тут Ваш код
```

poke4

запись полубайта в память

poke4 (`addr, val` **)**

Параметры:

`addr` - любой адрес из пространства 64Kb памяти, в который Вам нужно записать значение полубайта(4 бита)

`val` - значение которое нужно записать

Описание:

Эта функция позволяет записывать значения в память TIC - полубайт.

С её помощью можно, например, записывать значения пикселей в спрайты спрайт-листа.

Адрес указывается в шестнадцатеричном формате, значение в десятичном.

Стоит также отметить, что peek4 и poke4 оперируют полубайтами (4 бита), поэтому адрес умножается на два по отношению к обычным peek и poke, которые оперируют байтами (8 бит).

```
+----------+----------+----------+----------+
|   4bit   |   4bit   |   4bit   |   4bit   |
+----------+----------+----------+----------|
|         8bit        |         8bit        |
+---------------------+---------------------+
```

Пример:

Запустить или скачать картридж примера.

```
-- title:  poke4 demo
-- author: Al Rado
-- desc:   Set sprite from text to sprite sheet
-- script: lua
-- input:  gamepad
```

```
-- количество полубайт(nibble) в спрайте
NIBBLES_IN_SPR=8*8
-- адрес спрайт-листа в памяти, умножаю на два т.к. использую полубайты, а не байты
SPR_SHEET_ADDR=0x4000*2
-- индекс спрайта в спрайт-листе
sprIx=5
-- данные спрайта в шестнадцатеричном виде, каждый символ соответствует полубайту - индексу цвета(0-f)
-- можно получить, просто скопировав в буфер обмена в редакторе спрайтов
sprDataHex="a000000aa0f00f0aaaaaaaaaaafaaa6aafffaaaaaafaa6aaaaaaaaaa33333333"

cls()
for i=1,NIBBLES_IN_SPR do
  nibble=tonumber(sprDataHex:sub(i,i),16)
  poke4(SPR_SHEET_ADDR+(NIBBLES_IN_SPR*sprIx)+i-1,nibble)
end
sync()

print("See sprite sheet, index = "..sprIx)

function TIC() end
```

Пример 2:

```
-- script: lua
-- устанавливает значение пикселя в спрайт-листе

function sset(x,y,c)
  local addr=0x4000+(x//8+y//8*16)*32 -- получить адрес спрайта
  poke4(addr*2+x%8+y%8*8,c) -- установить значение пикселя
end

-- тут Ваш код
```

Запустить или скачать картридж примера.

```
-- title:  poke4 demo
-- author: Al Rado
-- desc:   Set sprite from text to sprite sheet
-- script: moon
-- input:  gamepad
```

```
-- количество полубайт(nibble) в спрайте
NIBBLES_IN_SPR=8*8
-- адрес спрайт-листа в памяти, умножаю на два т.к. использую полубайты, а не байты
SPR_SHEET_ADDR=0x4000*2
-- индекс спрайта в спрайт-листе
sprIx=5
-- данные спрайта в шестнадцатеричном виде, каждый символ соответствует полубайту - индексу цвета(0-f)
-- можно получить, просто скопировав в буфер обмена в редакторе спрайтов
sprDataHex="a000000aa0f00f0aaaaaaaaaaafaaa6aafffaaaaaafaa6aaaaaaaaaa33333333"

cls()
for i=1,NIBBLES_IN_SPR
  nibble=tonumber(sprDataHex\sub(i,i),16)
  poke4(SPR_SHEET_ADDR+(NIBBLES_IN_SPR*sprIx)+i-1,nibble)
sync()

print("See sprite sheet, index = "..sprIx)

export TIC=->
```

Пример 2:

```
-- script: moon
-- устанавливает значение пикселя в спрайт-листе

export sset=(x,y,c)->
 addr=0x4000+(x//8+y//8*16)*32 -- получить адрес спрайта
 poke4(addr*2+x%8+y%8*8,c) -- установить значение пикселя

-- тут Ваш код
```

```
// title: poke4 demo
// author: Al Rado
// desc: Set sprite from text to sprite sheet
// script: js
// input: gamepad

// количество полубайт(nibble) в спрайте
NIBBLES_IN_SPR = 8 * 8
// адрес спрайт-листа в памяти, умножаю на два т.к. использую полубайты, а не байты
SPR_SHEET_ADDR = 0x4000 * 2
// индекс спрайта в спрайт-листе
sprIx = 5
// данные спрайта в шестнадцатеричном виде, каждый символ соответствует полубайту - индексу цвета(0-f)
// можно получить, просто скопировав в буфер обмена в редакторе спрайтов
sprDataHex = "a000000aa0f00f0aaaaaaaaaaafaaa6aafffaaaaaafaa6aaaaaaaaaa33333333"

cls()
for (i = 0; i < NIBBLES_IN_SPR; i++) {
  nibble = parseInt(sprDataHex[i], 16)
  poke4(SPR_SHEET_ADDR + (NIBBLES_IN_SPR * sprIx) + i, nibble)
}
sync()

print("See sprite sheet, index = " + sprIx)

function TIC() { }
```

Пример 2:

```
// script: js
```

```
// устанавливает значение пикселя в спрайт-листе

function sset(x, y, c) {
  var addr = 0x4000 + (x / 8 + y / 8 * 16) * 32 // получить адрес спрайта
  poke4(addr * 2 + x % 8 + y % 8 * 8, c) // установить значение пикселя
}
```

memcpy

копирование байтов в памяти

memcpy (`dst, src, size` **)**

Параметры:

`dst` - адрес в памяти, куда будут записываться данные

`src` - адрес в памяти, откуда будут считываться данные

`size` - размер блока данных, в байта

Описание:

Копирует блок памяти указанного размера из одной области в другую.

Адрес указывается в шестнадцатеричном формате, но возвращаемое значение в десятичном.

Пример:

Чтобы скопировать спрайт #1 в спрайт #5 надо вызвать `memcpy(5*32+0x4000, 1*32+0x4000, 32)` , где 32 байта это память занимаемая спрайтом

memset

заполнение памяти указанным значением

memset (`dst, val, size` **)**

Параметры:

`dst` - адрес в памяти, с которого начнется заполнение

`val` - значение размером 1 байт, которым будет заполнена область

`size` - количество байт для записи

Описание:

Заполняет память указанным значением.

Адрес указывается в шестнадцатеричном формате, но возвращаемое значение в десятичном.

Пример:

Чтобы заполнить всю карту тайлом #2 надо вызвать `memset(0x8000, 2, 240*136)`

Значение `2` в данном контексте, означает индекс в карте тайлов.

Чтобы очистить экран цветом с индексом `0`: `memset(0x0000, 0, 240*136/2)`

Здесь размер делится на два (240*136/2) потому что цвет пикселя на экране занимает пол-байта, а не байт.

pmem

запись целого значения в постоянную память (persistent cart data)

pmem (`index, [val]` **) ->** `val`

Параметры:

`index` - индекс слота, в который Вы хотите записать/считать данные в постоянной памяти 0..6

`val` - значение, которое Вы хотите сохранить в памяти. Не указывайте этот параметр, если Вам нужно прочесть данные

Возвращает:

`val` - когда функция вызывается с единственным параметром, она возвращает значение хранящееся в слоте памяти.

Описание:

Эта функция позволяет сохранять и получать данные одного из 7 доступных слотов в постоянной памяти. Она полезна, чтобы сохранить таблицу достижений и любого рода продвижения.

Подсказка:

`pmem` зависит от хэш-суммы картриджа (md5), поэтому не меняйте данные картриджа, если вы хотите сохранить данные записанные с помощью `pmem`
Укажите уникальный tag savevid для вашей игры и его будет использовать `pmem` вместо того, чтобы полагаться на хеш MD5.

Пример:

Запустить или скачать картридж примера.

```
-- title:  pmem demo
-- author:
-- desc:
-- script: lua
-- input:  gamepad
-- saveid: "pmem demo"
```

```
-- load saved value at slot zero and save it back
-- incremented by 1
pmem(0,pmem(0)+1)

function TIC()
  cls()
  print("Started "..pmem(0).." times");
end
```

Запустить или скачать картридж примера.

```
-- title:  pmem demo
-- author:
-- desc:
-- script: moon
-- input:  gamepad
-- saveid: "pmem demo"

-- load saved value at slot zero and save it back
-- incremented by 1
pmem(0,pmem(0)+1)

export TIC=->
  cls()
  print "Started "..pmem(0).." times"
```

```
// title:  pmem demo
// author:
// desc:
// script: js
// input:  gamepad
// saveid: "pmem demo"

// load saved value at slot zero and save it back
// incremented by 1
pmem(0, pmem(0) + 1)

function TIC() {
  cls()
  print("Started " + pmem(0) + " times");
}
```

Звуки

Функции для работы со звуковыми эффектами и музыкой.

sfx

проигрывание звукового эффекта

sfx (`[id, [note, [duration=-1, [channel=0, [volume=15, [speed=0]]]]]]` **)**

Параметры: `id` - индекс эффекта в редакторе звуковых эффектов, от 0 до 63, -1 останов эффекта в указанном канале

`note` - нота, от 0 до 95, 12 нот в каждой из 8 октав, можно указать строковое значение ноты, например `C#4`

`duration` - длительность звучания, в тиках, по умолчанию равна бесконечности (-1)

`channel` - канал в котором будет воспроизводиться эффект, 0,1:квадрат, 2:треугольник и 3:шум

`volume` - громкость, от 0 до 15

`speed` - скорость воспроизведения, от -4 до 3

Описание:

Воспроизводит звуковой эффект по указанному id и параметрам. Для того чтобы остановить воспроизведение звукового эффекта, нужно указать id равный `-1` в том же канале.

Пример:

Примечание:

Для корректной работы данного примера Вам понадобится картридж.

Запустить или скачать картридж примера.

```
-- title:  sfx demo
-- author: Nesbox
-- desc:   how to play SFX demo
-- script: lua
-- input:  gamepad

t=0
id=0

function TIC()
```

```
  if btnp(0) then sfx(id,"E-4")
  elseif btnp(1) then sfx(id,"E-6")
  elseif btnp(2) then sfx(id,"E-7")
  elseif btnp(3) then sfx(id,"E-5")
  end

  cls(15)
  spr(1+(t%20)/10,102,24,1,4)
  print("PRESS ANY ARROW KEY",64,64,0)

  t=t+1

end
```

Запустить или скачать картридж примера.

```
-- title:  sfx demo
-- author: Nesbox
-- desc:   how to play SFX demo
-- script: moon
-- input:  gamepad

t=0
id=0

export TIC=->

  if btnp(0) then sfx(id,"E-4")
  elseif btnp(1) then sfx(id,"E-6")
  elseif btnp(2) then sfx(id,"E-7")
  elseif btnp(3) then sfx(id,"E-5")

  cls(15)
  spr(1+(t%20)/10,102,24,1,4)
  print("PRESS ANY ARROW KEY",64,64,0)

  t=t+1
```

```
// title:  sfx demo
// author: Nesbox
// desc:   how to play SFX demo
// script: js
// input:  gamepad

t = 0
id = 0

function TIC() {
  if (btnp(0)) { sfx(id, "E-4") }
  else if (btnp(1)) { sfx(id, "E-6") }
  else if (btnp(2)) { sfx(id, "E-7") }
  else if (btnp(3)) { sfx(id, "E-5") }

  cls(15)
  spr(1 + (t % 20) / 10, 102, 24, 1, 4)
  print("PRESS ANY ARROW KEY", 64, 64, 0)

  t++
}
```

music

проигрывание музыки

music (`[track=-1, [frame=-1, [loop=true]]]` **)**

Параметры:

`track` - индекс музыкального трека в музыкальном редакторе, от 0 до 15, -1 останов проигрывания музыки

`frame` - кадр, с которого начинать воспроизведение

`loop` - зациклено ли воспроизведение

Описание:

Воспроизводит музыкальный трек по указанному индексу трека. Для того чтобы остановить воспроизведение, нужно вызвать эту функцию без аргументов.

Другое

В данном разделе находятся функции API не вошедшие ни в одну из категорий.

time

возвращает количество тиков с момента старта игры

time () -> `ticks`

Возвращает:

`ticks` - количество миллисекунд прошедших с начала запуска приложения

Описание:

Функция возвращает количество миллисекунд прошедших с начала запуска приложения.
Полезно при отслеживании времени, анимации объектов и событиях изменяющихся во времени.

Пример:

Запустить или скачать картридж примера.

```
-- title:  time demo
-- author:
-- desc:
-- script: lua
-- input:  gamepad

function TIC()
  cls()
  --Show rising time
  print('Seconds elapsed: '..time()/1000)

  --Blink warning
  if(time()%500>250) then
    print('Warning!',0,30)
  end

  --After 2 seconds show this message
  if(time()>2000)then
    print('Fugit inreparabile tempus',0,60)
  end
end
```

Запустить или скачать картридж примера.

```
-- title:  time demo
-- author:
-- desc:
-- script: moon
-- input:  gamepad

export TIC=->
  cls()
  --Show rising time
  print 'Seconds elapsed: '..time()/1000

  --Blink warning
  if time()%500>250
    print 'Warning!',0,30

  --After 2 seconds show this message
  if time()>2000
    print 'Fugit inreparabile tempus',0,60
```

```
// title:  time demo
// author:
// desc:
// script: js
// input:  gamepad

function TIC() {
  cls()
  //Show rising time
  print('Seconds elapsed: ' + time() / 1000)

  //Blink warning
  if (time() % 500 > 250) {
    print('Warning!', 0, 30)
  }

  //After 2 seconds show this message
  if (time() > 2000) {
    print('Fugit inreparabile tempus', 0, 60)
  }
}
```

sync

сохранение изменений в спрайтах/карте тайлов во время игры

sync ()

Описание:
Данная функция используется для сохранения изменений в спрайтах/карте тайлов во время игры, иначе данные возвращаются к исходному состоянию.

exit

прерывает выполнение программы и производит выход в консоль

exit()

Описание:
Данная функция используется для выхода из программы в консоль. Точка выхода находится в конце тела функции TIC.

Список подключенных модулей

К **TIC-80** подключены некоторые стандартные модули Lua.
Язык Lua 5.3 собран в режиме совместимости с 5.2 (параметр `LUA_COMPAT_5_2`)
Прежде всего это заметно в модуле **math** - такие функции как: atan2, cosh, sinh, tanh, pow, frexp, ldexp оставлены из версии 5.2, в то время как из обычной версии Lua 5.3 их исключили.
Ниже представлен список библиотек и их функции/переменные в алфавитном порядке:

- **basic library**
  - assert (v [, message])
  - collectgarbage ([opt [, arg]])
  - dofile ([filename])
  - dump
  - error (message [, level])
  - _G
  - getmetatable (object)
  - ipairs (t)
  - next (table [, index])
  - pairs (t)
  - pcall (f [, arg1, ···])
  - rawequal (v1, v2)
  - rawget (table, index)
  - rawlen (v)
  - rawset (table, index, value)
  - select (index, ···)
  - setmetatable (table, metatable)
  - tonumber (e [, base])
  - tostring (v)
  - type (v)
  - _VERSION
  - xpcall (f, msgh [, arg1, ···])
- **coroutine library**
  - coroutine.create (f)
  - coroutine.isyieldable ()
  - coroutine.resume (co [, val1, ···])
  - coroutine.running ()
  - coroutine.status (co)
  - coroutine.wrap (f)
  - coroutine.yield (···)
- **string library**
  - string.byte (s [, i [, j]])
  - string.char (···)
  - string.dump (function [, strip])
  - string.find (s, pattern [, init [, plain]])
  - string.format (formatstring, ···)
  - string.gmatch (s, pattern)

- string.gsub (s, pattern, repl [, n])
- string.len (s)
- string.lower (s)
- string.match (s, pattern [, init])
- string.rep (s, n [, sep])
- string.reverse (s)
- string.sub (s, i [, j])
- string.upper (s)

- **table library**
  - table.concat (list [, sep [, i [, j]]])
  - table.insert (list, [pos,] value)
  - table.move (a1, f, e, t [,a2])
  - table.pack (⋯)
  - table.remove (list [, pos])
  - table.sort (list [, comp])
  - table.unpack (list [, i [, j]])

- **math library**
  - math.abs (x)
  - math.acos (x)
  - math.asin (x)
  - math.atan (y [, x])
  - math.atan2 (x, y)
  - math.ceil (x)
  - math.cos (x)
  - math.cosh (x)
  - math.deg (x)
  - math.exp (x)
  - math.floor (x)
  - math.fmod (x, y)
  - math.frexp (x)
  - math.huge
  - math.ldexp (m, e)
  - math.log (x [, base])
  - math.log10 (x)
  - math.max (x, ⋯)
  - math.maxinteger
  - math.min (x, ⋯)
  - math.mininteger
  - math.modf (x)
  - math.pi
  - math.rad (x)
  - math.random ([m [, n]])
  - math.randomseed (x)
  - math.sin (x)
  - math.sinh (x)
  - math.sqrt (x)
  - math.tan (x)

- math.tanh (x)
- math.tointeger (x)
- math.type (x)
- math.ult (m, n)

basic library

assert (`v [, message]` **)**

Вызывает error, если значение аргумента v = false (т.е., nil или false); иначе, возвращает все свои аргументы. В случае ошибки, message — это объект ошибки; когда он отсутствует, его значение по умолчанию «assertion failed!»

collectgarbage (`[opt [, arg]]` **)**

Эта функция общий интерфейс к сборщику мусора. Она выполняет различные действия в соответствии с её аргументом opt:

`«collect»` : выполняет полный цикл очистки мусора. Это опция по умолчанию.
`«stop»` : останавливает автоматическое выполнение сборщика мусора. Сборщик будет запущен, только когда явно вызван, до вызова его перезапуска.
`«restart»` : перезапускает автоматическое выполнение сборщика мусора.
`«count»` : возвращает общее количество используемой Lua памяти в килобайтах. Это значение имеет дробную часть, так что его произведение на 1024 дает точное количество байт, используемых Lua (за исключением переполнений).
`«step»` : выполняет шаг сборки мусора. «Размер» шага контролируется аргументом arg. С нулевым значением сборщик выполнит один базовый (неделимый) шаг. Для не нулевых значений, сборщик выполнит, как если это количество памяти (в килобайтах) было выделено Lua. Возвращает true, если шаг завершил цикл сборки.
`«setpause»` : устанавливает arg в качестве нового значения для паузы сборщика. Возвращает предыдущее значение паузы.
`«setstepmul»` : устанавливает arg в качестве нового значения для множителя шагов сборщика. Возвращает предыдущее значение шага.
`«isrunning»` : возвращает логическое значение, говорящее запущен ли сборщик (т.е. не остановлен).

dofile (`[filename]` **)**

Открывает файл и запускает его содержимое, как Lua кусок. Когда вызвана без аргументов, запускает содержимое стандартного ввода (stdin). Возвращает все значения, возвращенные куском. В случае ошибок, dofile распространяет ошибку вызывающему коду (т.е. dofile не запускается в защищенном режиме).

error (`message [, level]` **)**

Завершает последнюю защищенно-вызванную функцию и возвращает message, как объект ошибки. Функция error никогда не возвращается.

Обычно error добавляет информацию о позиции ошибки в начало сообщения, если сообщение (message) это строка. Аргумент level определяет как получить позицию ошибки. Когда level = 1 (по умолчанию) — позиция ошибки там, где вызвана функция error. Level = 2 — указывает ошибку там, где вызвана функция, вызвавшая error; и так далее. Передача level = 0 — позиция ошибки не включается в сообщение.

_G

Глобальная переменная (не функция), которая хранит глобальное окружение. Lua сама не использует эту переменную; её изменение не влияет на окружение, ни наоборот.

getmetatable (`object` **)**

Если object не имеет метатаблицы, возвращает nil. Иначе, если метатаблица объекта имеет поле "__metatable", возвращает ассоциированное с этим полем значение. Иначе, возвращает метатаблицу переданного объекта.

ipairs (`t` **)**

Возвращет три значения (функцию итератор, таблицу t и 0), так что конструкция `for i,v in ipairs(t) do body end` будет перебирать пары ключ–значение (1,t[1]), (2,t[2]), …, до первого значения nil.

next (`table [, index]` **)**

Позволяет программе просмостреть все поля таблицы. Её первый аргумент — это таблица, второй аргумент — индекс в этой таблице. nextвозвращает следующий индекс таблицы и ассоциированное с ним значение. Когда вызывается с nil в качестве второго аргумента, next возвращает начальный индекс и ассоциированное с ним значение. Когда вызывается с последним индексом, или с nil для пустой таблицы, next возвращает nil. Если второй аргумент отсутствует, то он интерпретируется как nil. В частности, вы можете использовать next(t) для проверки, что таблица пустая.

Порядок перечисления индексов не определен, даже для числовых индексов. (Для просмотра таблицы в числовом порядке, используйте числовой for.)

Поведение next неопределено, если во время просмотра вы присваиваете любое значение несуществующему полю в таблице. Тем не менее вы можете модифицировать существующие поля. В частности, вы можете очищать существующие поля.

pairs (`t` **)**

Если t имеет метаметод __pairs, вызывает его с аргументом t и возвращает первые три результата вызова.

Иначе, возвращает три значения: функцию next, таблицу t и nil, так что конструкция `for k,v in pairs(t) do body end` будет перебирать все пары ключ–значение в таблице t.

См. функцию next для предостережений о модификации таблицы во время просмотра.

pcall (`f [, arg1, ···]` **)**

Вызывает функцию f с переданными аргументами в защищенном режиме. Это значит, что любая ошибка внутри f не распространяется; взамен,pcall перехватывает ошибку и возвращает код статуса. Её первый результат — код статуса (логическое значение), которое равно true, если вызов успешен. В этом случае pcall также возвращает все результаты из вызова, после первого результата. В случае ошибки pcall возвращает false и сообщение об ошибке.

rawequal (`v1, v2` **)**

Проверяет, что v1 равен v2, без вызова метаметодов. Возвращает логическое значение.

rawget (`table, index` **)**

Возвращает реальное значение table[index], без вызова метаметодов. table должен быть таблицей, index может быть любым значением.

rawlen (`v` **)**

Возвращает длину объекта v, который должен быть таблицей или строкой, без вызова метаметодов. Возвращает целое.

rawset (`table, index, value` **)**

Устанавливает реальное значение table[index] равным value, без вызова метаметодов. table должен быть таблицей, index — любое значение отличное от nil и NaN, value — любое Lua значение.

Эта функция возвращает table.

select (`index, ···` **)**

Если index это число, возвращает все аргументы после аргумента номер index; негативное число индексируется с конца (-1 последний аргумент). Иначе, если index строка "#", select вернет общее количество дополнительных аргументов.

setmetatable (`table, metatable` **)**

Устанавливает метатаблицу для данной таблицы. (Вы не можете изменять метатаблицы других типов из Lua, только из C.) Если metatable = nil, удаляет метатаблицу переданной таблицы. Если оригинальная метатаблица имеет поле "__metatable", генерирует ошибку.
Эта функция возвращает table.

tonumber (`e [, base]` **)**

Когда вызвана без аргумента base, tonumber пытается конвертировать аргумент в число. Если аргумент уже число или строка, которую можно преобразовать в число, то tonumber возвращает это число; иначе, возвращает nil.

Преобразование строк может выдавать в результате целые или вещественные числа, в соответствии с лексическими соглашениями Lua. (Строка может иметь начальные и конечные пробелы и знак.)

Когда вызвана с аргументом base, то аргумент e должен быть строкой, которая интерпретируется как целое в данной системе счисления. base может быть любым целым от 2 до 36, включительно. При base > 10, символ ‘A‘ (в верхнем или нижнем регистре) представляет 10, ‘B‘ представляет 11 и так далее, с ‘Z‘ представляющим 35. Если строка e не правильное число в данной системе счисления, функция возвращает nil.

tostring (`v` **)**

Получает значение любого типа и преобразует его в строку в понятном человеку формате. (Для полного контроля над преобразованием чисел используйте string.format.)

Если метатаблица v имеет поле "__tostring", то tostring вызывает соответствующее значение с v в качестве аргумента и использует результат вызова, как свой результат.

type (`v` **)**

Возвращает тип любого аргумента, представленный строкой. Возможные результаты этой функции: «nil» (строка, не значение nil), «number», «string», «boolean», «table», «function», «thread» и «userdata».

_VERSION

Глобальная переменная (не функция), которая содержит строку с текущей версией интерпретатора. Текущее значение этой переменной «Lua 5.3».

xpcall (`f, msgh [, arg1, ···]` **)**

Эта функция похожа на pcall, но она устанавливает новый обработчик сообщений msgh.

coroutine library

coroutine.create (`f`)

Создает новый сопроцесс, с телом f. f должен быть функцией. Возвращает этот новый сопроцесс, как объект с типом "thread".

coroutine.isyieldable ()

Возвращает true, когда запущенный сопроцесс может уступить.

Запущенный сопроцесс может уступать, если это не главный поток и он не внутри неприостанавливаемой C функции.

coroutine.resume (`co [, val1, ···]`)

Начинает или продолжает выполнение сопроцесса co. При первом возобновлении сопроцесса запускает его тело. Значение val1, … передаются как аргументы телу сопроцесса (его функции). Если сопроцесс был приостановлен, resume перезапускает его; значниния val1, … передаются как результаты из yield.

Если сопроцесс запущен без ошибок, resume возвращает true и все значения, переданные в yield (когда сопроцесс уступает) или все значения, возвращенные функцией сопроцесса (когда сопроцесс завершается). В случае ошибок, resume возвращает false и сообщение об ошибке.

coroutine.running ()

Возвращает запущенный сопроцесс и логическое значение; true, если сопроцесс это главный поток.

coroutine.status (`co`)

Возвращает статус сопроцесса co, как строку: "running" — сопроцесс запущен (т.е. он вызвал status); "suspended" — сопроцесс приостановлен в вызове yield или еще не запущен; "normal" — сопроцесс активен, но не выполняется (т.е. он был продолжен другим сопроцессом); "dead" — сопроцесс завершил своё тело или был остановлен с ошибкой.

coroutine.wrap (`f`)

Создает новый сопроцесс с телом f. f должен быть функцией. Возвращает функцию, которая возобновляет сопроцесс при каждом её вызове. Все аргументы, переданные этой функции, ведут себя как дополнительные аргументы в resume. Возвращает те же значения, что и resume, за исключением первого логического значения. В случае ошибки, распространяет ошибку.

coroutine.yield (`···`)

Приостанавливает выполнение вызывающего сопроцесса. Все аргументы yield передаются, как дополнительные результаты в resume.

string library

string.byte (`s [, i [, j]]` **)**

Возвращает внутренние цифровые коды символов s[i], s[i+1], …, s[j]. По умолчанию i = 1; j = i. Эти индексы следуют тем же правилам, что и в функции string.sub.

Цифровые коды не обязательно портабельны между платформами.

string.char (`···` **)**

Получает ноль или более целых. Возвращает строку длиной равной количеству аргументов, в которой каждый символ имеет внутренний цифровой код равный соответствующему аргументу.

Цифровые коды не обязательно портабельны между платформами.

string.dump (`function [, strip]` **)**

Возвращает строку содержащую бинарное представление (бинарный кусок) переданной функции, так что load для этой строки возвращает копию функции (но с новыми upvalue). Если strip = true, бинарное представление может не включать всю отладочную информацию о функции, для уменьшения размера.

Функции с upvalue сохраняют только количество upvalue. При загрузке, эти upvalue получают свежие экземпляры, содержащие nil. (Вы можете использовать отладочную библиотеку, чтобы сохранить и перезагрузить upvalue функции в том виде, как вам нужно.)

string.find (`s, pattern [, init [, plain]]` **)**

Ищет первое совпадение шаблона pattern в строке s. Если совпадение найдено, то find возвращает индексы s, где совпадение начинается и заканчивается; иначе, возвращает nil. Третий опциональный цифровой аргумент init определяет, где начинать поиск; по умолчанию он равен 1 и может быть отрицательным. Значение true в качестве четвертого опционального аргумента plain выключает возможности поиска шаблонов, так функция выполняет плоский поиск подстроки, без магических символов в pattern. Учтите, что если передан plain, то должен быть передан и init.

Если шаблон имеет захваты (capture), то при успешном совпадении захваченные значения также возвращаются, после двух индексов.

string.format (`formatstring, ···` **)**

Возвращает форматированную версию переменного количества аргументов, следуя описанию в первом аргументе (должен быть строкой). formatstring — следует тем же правилам, что и в функции sprintf в ISO C. Только отличается тем, что опции/модификаторы *, h, L, l, n и p не поддерживаются, и тем, что имеет дополнительную опцию q. Опция q форматирует строку между двойными кавычками и использует управляющие символы, когда необходимо гарантировать, что строка может быть прочитана Lua интерпретатором обратно. Например, вызов

```
string.format('%q', 'a string with "quotes" and \n new line')
```

может выдать строку:

```
"a string with \"quotes\" and \
 new line"
```

Опции A, a, E, e, f, G и g — все ожидают цифровой аргумент. Опции c, d, i, o, u, X и x — ожидают целое. Опция q — ожидает строку. Опция s — ожидает строку без встроенных нулей; если аргумент не строка, он конвертируется следуя тем же правилам, что и в tostring.

Когда Lua скомпилирована с не C99 компилятором, опции A и a (шестнадцатиричные вещественные числа) не поддерживают модификаторы (флаги, ширина, длина).

string.gmatch (`s, pattern` **)**

Возвращает функцию-итератор, которая при каждом вызове возвращает следующие захваченные значения из pattern по строке s. Если pattern не определяет захватов, то в каждом вызове возвращается целое совпадение.

Например, следующий цикл будет перебирать все слова из строки s, печатая по одному в строке:

```
s = "hello world from Lua"
for w in string.gmatch(s, "%a+") do
  print(w)
end
```

Следующий пример собирает в таблице все пары key=value из строки:

```
t = {}
s = "from=world, to=Lua"
for k, v in string.gmatch(s, "(%w+)=(%w+)") do
  t[k] = v
end
```

Для этой функции, символ ‘^‘ в начале шаблона не работает как якорь, т.к. это мешает итерации.

string.gsub (`s, pattern, repl [, n]` **)**

Возвращает копию s, в которой все (или первые n, если передано) совпадения шаблона pattern заменены на замещающую строку, определенную параметром repl, который может быть строкой, таблицей или функцией. gsub также возвращает общее число совпадений, как второе значение. Имя gsub происходит от Global SUBstitution (глобальная подстановка).

Если repl это строка, то её значение используется для замены. Символ % работает, как управляющий символ: любая последовательность в repl в виде %d, с d между 1 и 9, соответствует d-ой захваченной подстроке. Последовательность %0 соответствует полному совпадению. Последовательность %% соответствует одному символу %.

Если repl это таблица, то таблица запрашивается для каждого совпадения, используя первое захваченное значение, как ключ.

Если repl это функция, то эта функция вызывается для каждого совпадения, все захваченные подстроки передаются в качестве аргументов, по порядку.

В любом случае, если шаблон не имеет захватов, то он ведет себя так, будто весь шаблон находится в захвате.

Если значение, возвращенное из табличного запроса или из функции, это строка или число, то оно используется, как замещающая строка; иначе, если это false или nil, то замена не производится (т.е. оригинальное содержимое совпадения сохраняется в строке).

Несколько примеров:

```
x = string.gsub("hellod", "(%w+)", "%1 %1")
--> x="hello hello world world"

x = string.gsub("hellod", "%w+", "%0 %0", 1)
--> x="hello hello world"

x = string.gsub("hellod from Lua", "(%w+)%s*(%w+)", "%2 %1")
--> x="world hello Lua from"

x = string.gsub("homeOME, user = $USER", "%$(%w+)", os.getenv)
--> x="home = /home/roberto, user = roberto"

x = string.gsub("4+5eturn 4+5$", "%$(.-)%$", function (s)
      return load(s)()
    end)
--> x="4+5 = 9"

local t = {name="lua", version="5.3"}
x = string.gsub("$name-$version.tar.gz$(%w+)", t)
--> x="lua-5.3.tar.gz"
```

string.len (`s` **)**

Получает строку и возвращает её длину. Пустая строка "" имеет длину 0. Встроенные нули считаются, так "a\000bc\000" имеет длину 5.

string.lower (`s` **)**

Получает строку и возвращает её копию с заменой всех символов в верхнем регистре на символы в нижнем регистре. Остальные символы не изменяются. Определение какие символы в верхнем регистре зависит от текущей локали.

string.match (`s, pattern [, init]` **)**

Ищет первое совпадение шаблона pattern в строке s. Если находит, то match возвращает захваченные значения из шаблона; иначе возвращает nil. Если pattern не описывает захватов, то возвращается целое совпадение. Третий опциональный цифровой аргумент init определяет, где начинать поиск; по умолчанию он равен 1, и может быть отрицательным.

string.rep (`s, n [, sep]` **)**

Возвращает строку, которая состоит из слияния n копий строки s, разделенных строкой sep. Значение по умолчанию для sep это пустая строка (т.е. нет разделителя). Возвращает пустую строку, если n отрицательное значение.

string.reverse (`s` **)**

Возвращает строку, в которой символы s идут в обратном порядке.

string.sub (`s, i [, j]` **)**

Возвращает подстроку s, начинающуюся на i и продолжающуюся до j; i и j могут быть отрицательными. Если j отсутствует, то он подразумевается равным -1 (тоже что и длина строки). В частности, вызов string.sub(s,1,j) возвращает префикс s длиной j, и string.sub(s, -i) возвращает суффикс s длиной i.

Если после трансляции отрицательных индексов i < 1, он корректируется до 1. Если j больше длины строки, он корректируется до этой длины. Если после этих преобразований i > j, функция возвращает пустую строку.

string.upper (`s` **)**

Получает строку и возвращает её копию с заменой всех символов в нижнем регистре на символы в верхнем регистре. Остальные символы не изменяются. Определение какие символы в нижнем регистре зависит от текущей локали.

table library

table.concat (`list [, sep [, i [, j]]]` **)**

Получает список list, где все элементы строки или числа, возвращает строку list[i]..sep..list[i+1] ··· sep..list[j]. По умолчанию, sep это пустая строка, i = 1 и j = #list. Если i > j, возвращает пустую строку.

table.insert (`list, [pos,] value` **)**

Вставляет элемент value на позицию pos в список list, сдвигая элементы вверх list[pos], list[pos+1], ···, list[#list]. По умолчанию,pos = #list+1, так вызов table.insert(t,x) вставляет x в конец списка t.

table.move (`a1, f, e, t [,a2]` **)**

Перемещает элементы из таблицы a1 в таблицу a2. Эта функция эквивалентна множественному присваиванию: a2[t],··· = a1[f],···,a1[e]. По умолчанию, a2 = a1. Целевой диапазон может перекрываться с диапазоном источником. Количество элементов для перемещения должно помещаться в целое Lua.

table.pack (`···` **)**

Возвращает новую таблицу, в которой все параметры сохранены с ключами 1, 2 и т.д. и поле «n» содержит количество параметров. Учтите, что результирующая таблица может не быть последовательностью.

table.remove (`list [, pos]` **)**

Удаляет из списка list элемент на позиции pos, возвращая значение этого элемента. Когда pos это целое между 1 и #list, функция сдвигает элементы вниз list[pos+1], list[pos+2], ···, list[#list] и стирает элемент list[#list]; Индекс pos может быть 0, когда #list = 0, или#list + 1; в этих случаях функция стирает элемент list[pos].

По умолчанию, pos = #list, так вызов table.remove(l) удаляет последний элемент списка l.

table.sort (`list [, comp]` **)**

Сортирует элементы полученного списка, на месте, от list[1] до list[#list]. Если передан параметр comp, он должен быть функцией, которая получает два элемента списка и возвращает true, когда первый элемент должен находиться перед вторым в финальном упорядочении (так выражение not comp(list[i+1],list[i]) будет истинным после сортировки). Если параметр comp не передан, то взамен Lua использует стандартный оператор <.

Алгоритм сортировки не стабилен; т.е. элементы, считающиеся равными в этом упорядочении, в результате сортировки могут изменить свои относительные позиции.

table.unpack (`list [, i [, j]]` **)**

Возвращает элементы из полученного списка. Эта функция эквивалентна

```
return list[i], list[i+1], ···, list[j]
```

По умолчанию, i = 1 и j = #list.

math library

math.abs (`x`)

Возвращает абсолютное значение x. (integer/float)

math.acos (`x`)

Возвращает арккосинус x (в радианах).

math.asin (`x`)

Возвращает арксинус x (в радианах).

math.atan (`y [, x]`)

Возвращает арктангенс y/x (в радианах), но использует знаки обоих параметров для поиска квадранта результата. (Также корректно обрабатывает случай, когда x = 0.)

По умолчанию x = 1, так вызов math.atan(y) возвращает арктангенс y.

math.atan2 (`x, y`)

Возвращает арктангенс x/y (в радианах), но использует знаки обоих параметров для вычисления «четверти» на плоскости. (Также корректно обрабатывает случай когда y равен нулю.)

math.ceil (`x`)

Возвращает наименьшее целое значение, которое больше или равно x.

math.cos (`x`)

Возвращает косинус x (в радианах).

math.cosh (`x`)

Возвращает кошинус (гиперболический косинус) x.

math.deg (`x`)

Преобразует угол x из радиан в градусы.

math.exp (`x`)

Возвращает значение ex (где e — основание натурального логарифма).

math.floor (`x`)

Возвращает наибольшее значение, которое меньше или равно x.

math.fmod (`x, y`)

Возвращает остаток от деления x на y, который округляет частное к нулю. (integer/float)

math.frexp (`x`)

Возвращает m и e такие, что x = m2e, e – целое, а модуль m находится в интервале [0.5, 1) (либо ноль, если x равен нулю). (Разложение числа с фиксированной запятой).

math.huge

Вещественное значение HUGE_VAL, которое больше любого другого числового значения.

math.ldexp (`m, e`)

Возвращает m2e (e должно быть целым). (Восстановление значения по мантиссе и показателю).

math.log (`x [, base]`)

Возвращает логарифм x по основанию base. По умолчанию, base = e (так функция возвращает натуральный логарифм x).

math.log10 (`x`)

Возвращает логарифм x по основанию 10.

math.max (`x, ···`)

Возвращает аргумент с максимальным значением, в соответствии с Lua оператором <. (integer/float)

math.maxinteger

Целое с максимальным значением для целого.

math.min (`x, ···`)

Возвращает аргумент с минимальным значением, в соответствии с Lua оператором <. (integer/float)

math.mininteger

Целое с минимальным значением для целого.

math.modf (`x`)

Возвращает целую и дробную часть x. Второй результат всегда вещественное число.

math.pi

Значение π.

math.rad (`x`)

Преобразует угол x из градусов в радианы.

math.random (`[m [, n]]`)

Когда вызвана без аргументов, возвращает псевдослучайное вещественное число с однородным распределением в диапазоне [0,1). Когда вызвана с двумя целыми m и n, math.random возвращает псевдослучайное целое с однородным распределением в диапазоне [m, n]. (Значение m-n не может быть отрицательным и должно помещаться в целое Lua.) Вызов math.random(n) эквивалентен вызову math.random(1,n).

Эта функция является интерфейсом к генератору псевдослучайных чисел, предоставляемому C. Нет гарантий для его статистических свойств.

math.randomseed (`x`)

Устанавливает x как «затравку» (seed) для генератора псевдослучайных чисел: одинаковые затравки производят одинаковые последовательности чисел.

math.sin (`x`)

Возвращает синус x (в радианах).

math.sinh (`x`)

Возвращает шинус (гиперболический синус) x.

math.sqrt (`x`)

Возвращает квадратный корень x. (Для вычисления этого значения вы также можете использовать выражение x^0.5.)

math.tan (`x`)

Возвращает тангенс x (в радианах).

math.tanh (`x`)

Возвращает гиперболический тангенс x.

math.tointeger (`x`)

Если значение x можно преобразовать в целое, возвращает целое. Иначе, возвращает nil.

math.type (`x`)

Возвращает «integer» — если x целое, «float» — если x вещественное, или nil — если x не число.

math.ult (`m, n`)

Возвращает логическое значение, true, если целое m ниже целого n, когда они сравниваются как беззнаковые целые.

Изучаем Lua за 15 минут

Перевод: Al Rado - Twitter @alrado2, Никита Курылев Twitter @nikita_kurilev
Оригинал статьи на английском тут.

```
-- Два тире - начинают однострочный комментарий.

--[[
     Если добавить два знака "[" и "]", то получится
     многострочный комментарий
--]]

----------------------------------------------------
-- 1. Переменные и управление процессами.
----------------------------------------------------

num = 42  -- Все числа - числа с плавающей запятой.
-- Не беспокойтесь, у 64-битных чисел с плавающей запятой есть 52 бита чтобы
-- хранить точные int значения; машинная точность не
-- является проблемой для int'ов которым нужно меньше 52 бит.

s = 'walternate'  -- Неизменные строки как в Python'е.
t = "Двойные кавычки тоже работают"
u = [[ Двойные квадратные скобки
       начинают и заканчивают
       многострочный текст.]]
t = nil  -- переменная t неопределена; У Lua присутствует сборщик мусора.

-- Блоки обозначаются с помощью "do" и "end":
while num < 50 do
  num = num + 1  -- Нет таких операторов как "++" или "+=".
end

-- Условия "Если":
if num > 40 then
  print('over 40')
elseif s ~= 'walternate' then  -- "~=" значит "не равно".
  -- Для проверки равенства используется "==" как в Python'е; подходит и для строк.
  io.write('not over 40\n')  -- По-умолчанию вывод на stdout.
else
  -- Переменные глобальны по умолчанию.
  thisIsGlobal = 5  -- Общепринят кэмелКейс.

  -- Как сделать переменную локальной:
  local line = io.read()  -- читает следующую строку из стандартного потока ввода stdin.

  -- Для объединения строк используется "..":
  print('Winter is coming, ' .. line)
end

-- Неопределённые переменные возвращают "nil".
-- Это не ошибка:
foo = anUnknownVariable  -- Теперь foo равняется nil.

aBoolValue = false

-- Только "nil" и "false" возвращают 'ложно'; "0" и '' возвращают 'истинно'!
if not aBoolValue then print('twas false') end

-- 'or' и 'and' используются для упрощения.
```

```lua
-- Это схоже с тернарным оператором "a?b:c" в C/js:
ans = aBoolValue and 'yes' or 'no'  --> 'no'

karlSum = 0
for i = 1, 100 do  -- В этот диапазон включены оба значения.
  karlSum = karlSum + i
end

-- Используйте "100, 1, -1" как диапазон чтобы сделать обратный отсчёт:
fredSum = 0
for j = 100, 1, -1 do fredSum = fredSum + j end

-- В общем случае, для диапазона указывается начало, конец[, шаг].

-- Другая конструкция цикла:
repeat
  print('the way of the future')
  num = num - 1
until num == 0


----------------------------------------------------
-- 2. Функции.
----------------------------------------------------

function fib(n)
  if n < 2 then return 1 end
  return fib(n - 2) + fib(n - 1)
end

-- Замыкания и анонимные функции работают:
function adder(x)
  -- Возвращаемая функция создаётся, когда "adder"
  -- вызывается, и запоминает значение x:
  return function (y) return x + y end
end
a1 = adder(9)
a2 = adder(36)
print(a1(16))  --> 25
print(a2(64))  --> 100

-- Возвраты, вызовы функций, и назначения работают
-- со списками, которые могут несовпадать по длине.
-- Несовпадающие получатели выставляются nil'ами;
-- несовпадающие отправители отбрасываются.

x, y, z = 1, 2, 3, 4
-- Тут x = 1, y = 2, z = 3, а 4 отброшен.

function bar(a, b, c)
  print(a, b, c)
  return 4, 8, 15, 16, 23, 42
end

x, y = bar('zaphod')  --> выводит "zaphod  nil nil"
-- Тут x = 4, y = 8, значения 15..42 отброшены.

-- Функции являются значениями первого класса, могут быть локальными/глобальными.
-- Эти функции одинаковы:
function f(x) return x * x end
f = function (x) return x * x end

-- Также как и эти:
local function g(x) return math.sin(x) end
local g; g  = function (x) return math.sin(x) end
-- объявление 'local g' создает переменную, затем производится присвоение
```

```lua
-- Кстати, тригонометрические функции работают в радианах.

-- Вызовы с одним строковым параметром не нуждаются в скобках:
print 'hello'  -- Вполне работает.


----------------------------------------------------
-- 3. Таблицы.
----------------------------------------------------

-- Таблица = единственная составная структура данных Lua;
--          является ассоциативным массивом.
-- Подобно масивам в php или объектам в js, они являются
-- словарями с поиском по хэшу, которые могут быть использованы как списки

-- Использование таблиц в качестве словарей / карт:

-- Элементы словарей имеют строковые ключи по умолчанию:
t = {key1 = 'value1', key2 = false}

-- Для доступа к строковым ключам может использоваться js-подобная точечная нотация:
print(t.key1)  -- Печатает 'value1'.
t.newKey = {}  -- Добавляет новую пару ключ/значение.
t.key2 = nil   -- Удаляет key2 из таблицы.

-- Литеральная нотация для любых(не nil) значений как ключ:
u = {['@!#'] = 'qbert', [{}] = 1729, [6.28] = 'tau'}
print(u[6.28])  -- выводит "tau"

-- Ключи задаются в основном числами
-- и строками, но возможна идентификация таблицами.
a = u['@!#']  -- Тут a = 'qbert'.
b = u[{}]     -- Мы могли бы ожидать 1729, но он равен nil:
-- b = nil поскольку поиск не удался. Он не удался
-- потому что ключ который мы использовали не тот же самый объект
-- который был использован для сохранения оригинального значения.
-- Так что строки и числа являются более переносимыми ключами.

-- при вызове функции с параметром-таблицей не нужны круглые скобки:
function h(x) print(x.key1) end
h{key1 = 'Sonmi~451'}  -- Prints 'Sonmi~451'.

for key, val in pairs(u) do  -- Итерация таблицы.
  print(key, val)
end

-- _G представляет собой специальную таблицу всех глобальных переменных.
print(_G['_G'] == _G)  -- Печатает 'true'.

-- Использование таблиц как списки/массивы:

-- Список литералов неявно задает int-овые ключи:
v = {'value1', 'value2', 1.21, 'gigawatts'}
for i = 1, #v do  -- здесь #v это размер списка v
  print(v[i])  -- Индексы начинаются с 1!! Сумашедшие чтоли! )
end
-- 'список' не является реальным типом. v это таблица
-- с последовательными целыми ключами, обрабатываемыми как список

----------------------------------------------------
-- 3.1 Метатаблицы и метаметоды.
----------------------------------------------------

-- Таблица может иметь метатаблицу, который дает таблицу
-- оператор-перегруженное поведение. Далее вы увидете
```

```lua
-- как метатаблицы поддерживают поведение js-прототипа.

f1 = {a = 1, b = 2}  -- Представляет фракцию a/b.
f2 = {a = 2, b = 3}

-- Это выдаст ошибку:
-- s = f1 + f2

metafraction = {}
function metafraction.__add(f1, f2)
  sum = {}
  sum.b = f1.b * f2.b
  sum.a = f1.a * f2.b + f2.a * f1.b
  return sum
end

setmetatable(f1, metafraction)
setmetatable(f2, metafraction)

s = f1 + f2  -- вызывается __add(f1, f2) для метатаблицы f1

-- f1, f2 не имеют ключа для их метатаблиц, в отличии от
- прототипв в js, поэтому вы должны получить его, как в
- getmetatable (f1). Метатаблица является обычной таблицей
- с ключами, о которых знает Lua, например __add.


-- Но следующая строка выдаст ошибку, так как s не имеет метаданных:
-- t = s + s
-- Классовые шаблоны, приведенные ниже, исправят это.

-- __index для метаданных перегружает точки просмотра:
defaultFavs = {animal = 'gru', food = 'donuts'}
myFavs = {food = 'pizza'}
setmetatable(myFavs, {__index = defaultFavs})
eatenBy = myFavs.animal  -- работает! Благодаря метатаблице

-- Прямой поиск таблиц, которые не сработают, повторит попытку
-- значение __index метатаблицы, и это рекурсивно.

-- Значение __index также может быть функцией (tbl, key)
-- для более определенного поиска.

-- Значения __index, add, .. называются метаметодами.
-- Полный список. Вот таблица с метаметодами.

-- __add(a, b)                     for a + b
-- __sub(a, b)                     for a - b
-- __mul(a, b)                     for a * b
-- __div(a, b)                     for a / b
-- __mod(a, b)                     for a % b
-- __pow(a, b)                     for a ^ b
-- __unm(a)                        for -a
-- __concat(a, b)                  for a .. b
-- __len(a)                        for #a
-- __eq(a, b)                      for a == b
-- __lt(a, b)                      for a < b
-- __le(a, b)                      for a <= b
-- __index(a, b)  <fn or a table>  for a.b
-- __newindex(a, b, c)             for a.b = c
-- __call(a, ...)                  for a(...)

-----------------------------------------------------
-- 3.2 Классо-подобные таблицы и наследование.
-----------------------------------------------------
```

```lua
-- Классы не встроены; Есть разные способы
-- сделать их с использованием таблиц и метаданных.

-- Объяснение для этого примера ниже.

Dog = {}                                  -- 1.

function Dog:new()                        -- 2.
  newObj = {sound = 'woof'}               -- 3.
  self.__index = self                     -- 4.
  return setmetatable(newObj, self)       -- 5.
end

function Dog:makeSound()                  -- 6.
  print('I say ' .. self.sound)
end

mrDog = Dog:new()                         -- 7.
mrDog:makeSound()  -- 'I say woof'        -- 8.

-- 1. Dog работает как класс; В действительности это таблица.
-- 2. function tablename:fn(...) так же как
--    function tablename.fn(self, ...)
--    Просто добавляет первый аргумент, называемый self.
--    Прочитайте 7 и 8 ниже о том, как 'self' получает свое значение.
-- 3. newObj будет экземпляром класса Dog.
-- 4. self = экземпляр класса. Часто
--    self = Dog, но наследование может изменить это.
--    NewObj получает функции self, когда мы устанавливаем оба значения:
--    метатаблицу newObj и __index self-а для себя.
-- 5. Напоминание: setmetatable возвращает свой первый аргумент.
-- 6. Работает как в пункте 2, но на этот раз мы ожидаем что
--    self будет экземпляром вместо класса.
-- 7. То же, что и Dog.new (Dog), поэтому self = Dog в new ().
-- 8. То же, что mrDog.makeSound (mrDog); Self = mrDog.

----------------------------------------------------

-- Пример наследования:

LoudDog = Dog:new()                           -- 1.

function LoudDog:makeSound()
  s = self.sound .. ' '                       -- 2.
  print(s .. s .. s)
end

seymour = LoudDog:new()                       -- 3.
seymour:makeSound()  -- 'woof woof woof'      -- 4.

-- 1. LoudDog получает методы и переменные Dog.
-- 2. self имеет «звуковой» ключ после применения new (), см. 3.
-- 3.То же, что LoudDog.new (LoudDog), и преобразуется в
-- Dog.new (LoudDog), поскольку LoudDog не имеет ключа 'new',
-- но у него есть __index = Dog в его метатаблице.
-- Результат:  метатаблица seymour это LoudDog, и
-- LoudDog.__ index = LoudDog. Так что seymour.key будет
-- равен seymour.key, LoudDog.key, Dog.key, какая
-- таблица является первой с данным ключом
-- 4. Ключ 'makeSound' находится в LoudDog;
-- это то же самое, что и LoudDog.makeSound (seymour)

-- При необходимости, создание подкласса может быть похоже на создание базового:
function LoudDog:new()
  newObj = {}
  -- set up newObj
```

```lua
  self.__index = self
  return setmetatable(newObj, self)
end

----------------------------------------------------
-- 4. Модули.
----------------------------------------------------


--[[ Я комментирую этот раздел, так что остальная часть
-- этого сценария остается работоспособной.
-- Предположим, файл mod.lua выглядит так:
local M = {}

local function sayMyName()
  print('Hrunkner')
end

function M.sayHello()
  print('Why hello there')
  sayMyName()
end

return M

-- Другой файл может использовать функциональность mod.lua:
local mod = require('mod')  -- Запуск файла mod.lua.

-- require - стандартный способ подключения модулей.
-- require работает так: (если модули не кэшированы, см. Ниже)
local mod = (function ()
  <contents of mod.lua>
end)()
-- Можно представить mod.lua как тело функции, так что
-- локальные переменные внутри mod.lua невидимы вне его.

-- Это работает, потому что mod здесь = M в mod.lua:
mod.sayHello()  -- Приветствует Hrunkner'a.

-- Это не верно; sayMyName существует только в mod.lua:
mod.sayMyName()  -- ошибка

-- require кэширует возвращаемые значения, поэтому файл
-- запускается не более одного раза, даже когда require вызывается много раз.

-- Предположим mod2.lua содержит "print('Hi!')".
local a = require('mod2')  -- Печатает Hi!
local b = require('mod2')  -- Не печатает; a=b.

-- dofile похож на require но без кэширования:
dofile('mod2.lua')  --> Hi!
dofile('mod2.lua')  --> Hi! (runs it again)

-- loadfile загружает lua файл, но не запускает его
f = loadfile('mod2.lua')  -- Нужно вызвать f() для его запуска.

-- Loadstring - это loadfile для строк.
g = loadstring('print(343)')  -- Возвращает функцию.
g()  -- Выводит на печать 343; Ничего не было напечатано до сих пор.

--]]

----------------------------------------------------
-- 5. Ссылки.
----------------------------------------------------
```

```
--[[

Я был рад узнать Lua, чтобы я мог делать игры
на игровом движке Löve 2D. Вот почему.

Я начал с "BlackBulletIV Lua" для программистов.
Затем я прочитал официальную книгу "Программирование на языке Lua".
Вот как.

Может быть полезно пройти по
ссылке на lua-users.org.

Основные темы, которые не были охвачены - стандартные библиотеки:
 * string library
 * table library
 * math library
 * io library
 * os library

Кстати, весь этот файл работоспособен на Lua; сохрани это
как learn.lua и запусти его [из командной строки] "lua learn.lua"!

Это было впервые написано для tylerneylon.com. Это
также доступно на github gist. Учебники для других
языков в том же стиле, что и этот, здесь:

http://learnxinyminutes.com/

Приятной работы с Луа!

--]]
```

Изучаем MoonScript за 15 минут

Перевод: Михаил Радюк - Twitter @torabora08
Оригинал статьи на английском тут.

```
-- Два тире начинают комментарий. Комментарии могут продолжаться до конца строки.
-- MoonScript скомпилированный в Lua не содержит комментариев.

-- Примечание: в MoonScript не используются 'do', 'then', или 'end' как в Lua,
-- вместо этого используется синтаксис с отступом, скорее похожий на Python.

----------------------------------------------------
-- 1. Присваивание
----------------------------------------------------

hello = "world"
a, b, c = 1, 2, 3
hello = 123 -- Перезаписывает `hello`, который выше.

x = 0
x += 10 -- x = x + 10

s = "hello "
s ..= "world" -- s = s .. "world"

b = false
b and= true or false -- b = b and (true or false)

----------------------------------------------------
-- 2. Литералы и операторы
----------------------------------------------------

-- Литералы работают практически также как и в Lua. Строки могут быть
-- разорваны без использования знака \.

some_string = "exa
mple" -- local some_string = "exa\mple"

-- Строки также могут включать интерполированные значения или значения,
-- которые определяются и потом помещаются внуть строки

some_string = "This is an #{some_string}" -- Превращается в 'This is an exa\mple'

----------------------------------------------------
-- 2.1. Литералы функций
----------------------------------------------------

-- При записи функций используются стрелки:

my_function = -> -- компилируется в `function() end`
my_function() -- вызывает пустую функцию

-- Функции могут вызываться без использования скобок. Скобки
-- могут быть использованы для приоритета над другими функциями

func_a = -> print "Hello World!"
func_b = ->
    value = 100
    print "The value: #{value}"

-- Если функция не требует параметров, она может быть вызвана либо с помощью `()` либо с `!`.
```

```
func_a!
func_b()

-- Функции могут использовать аргументы перед стрелкой, указанные списком
-- имён аргументов, заключенных в скобки.

sum = (x, y)-> x + y -- Из функции возвращается крайнее выражение.
print sum(5, 10)

-- В Lua есть идиома передачи первого аргумента в функцию как объекта,
-- в качестве объекта "собственного" объекта. Использование толстой стрелки (=>) вместо тонкой (->)
-- автоматически создаёт переменную "себя". `@x` это сокращенная запись `self.x`.

func = (num)=> @value + num

-- Аргументы по-умолчанию могут быть также использованы с литералами функций:

a_function = (name = "something", height=100)->
    print "Hello, I am #{name}.\nMy height is #{height}."

-- От того, что аргументы по-умолчанию вычисляются в теле функции во время компиляции в Lua,
-- вы можете ссылаться на предыдущие аргументы

some_args = (x = 100, y = x + 1000)-> print(x + y)

-----------------------------------------------------
-- Некоторые аспекты
-----------------------------------------------------

-- Знак минуса играет две роли: унарного оператора отрицания и бинарного
-- оператора вычитания. Рекомендуется всегда использовать пробелы между
-- бинарными операторами во избежание коллизий.

a = x - 10 --  a = x - 10
b = x-10 -- b = x - 10
c = x -y -- c = x(-y)
d = x- z -- d = x - z

-- Когда нет пробела между переменной и строковым литералом
-- вызов функции берет приоритет над последующим выражением:

x = func"hello" + 100 -- func("hello") + 100
y = func "hello" + 100 -- func("hello" + 100)

-- Аргументы в функции могут располагаться на нескольких строках в зависимости от того,
-- насколько аргументы выделены отступами. Отступы также могут быть вложенными.

my_func 5, -- вызывается как my_func(5, 8, another_func(6, 7, 9, 1, 2), 5, 4)
    8, another_func 6, 7, -- вызывается как
        9, 1, 2,          -- another_func(6, 7, 9, 1, 2)
    5, 4

 -- Если вызов функции стоит в начале блока, то выделение отступами может отличаться
 -- от уровня отступов, используемых в блоке

if func 1, 2, 3, -- called as func(1, 2, 3, "hello", "world")
        "hello",
        "world"
    print "hello"

-----------------------------------------------------
-- 3. Таблицы
-----------------------------------------------------

-- Таблицы определяются фигурными скобками, как в Lua:
```

```
some_values = {1, 2, 3, 4}

-- В таблицах можно использовать новую строку вместо запятых.

some_other_values = {
    5, 6
    7, 8
}

-- Назначение выполняется через `:` вместо `=`:

profile = {
    name: "Bill"
    age: 200
    "favorite food": "rice"
}

-- Фигурные скобки могут быть опущены для таблиц типа `ключ: значение`.

y = type: "dog", legs: 4, tails: 1

profile =
    height: "4 feet",
    shoe_size: 13,
    favorite_foods: -- вложенная таблица
        foo: "ice cream",
        bar: "donuts"

my_function dance: "Tango", partner: "none" -- :( вечно одинок

-- Таблицы, построенные из переменных могут использовать те же имена
-- используя `:` как префиксный оператор.

hair = "golden"
height = 200
person = {:hair, :height}

-- Как и в Lua, ключи могут быть нестрочными и нечисловыми значениями при применении `[]`.

t =
    [1 + 2]: "hello"
    "hello world": true -- Можно использовать строчные литералы без `[]`.

-----------------------------------------------------
-- 3.1. Табличные генераторы
-----------------------------------------------------

-- Генераторы списков

-- Создаётся копия списка, но с удвоением всех элементов. Использование
-- звёздочки перед именем переменной или таблицы применяется для перебора значений таблицы.

items = {1, 2, 3, 4}
doubled = [item * 2 for item in *items]
-- -- `when` используется когда переменная должна быть включена

slice = [item for item in *items when i > 1 and i < 3]

-- Конструкции `for` внутри списочных генераторов могут быть соединены

x_coords = {4, 5, 6, 7}
y_coords = {9, 2, 3}

points = [{x,y} for x in *x_coords for y in *y_coords]
```

```
-- Числовые циклы for также могут использоваться в генераторах:

evens = [i for i=1, 100 when i % 2 == 0]

-- Табличные генераторы очень похожи, но используют `{` и `}`
-- и берут два значения для каждой итерации.

thing = color: "red", name: "thing", width: 123
thing_copy = {k, v for k, v in pairs thing}

-- Таблицы могут быть сделаны "плоскими" из пар "ключ-значение" в массиве при помощи `unpack`
-- для возврата обоих значений, используя первое как ключ и второе как значение.

tuples = {{"hello", "world"}, {"foo", "bar"}}
table = {unpack tuple for tuple in *tuples}

-- Слайсинг (slicing) выполняется чтобы перебрать только определенную часть массива.
-- Для перебора используется нотация `*`, но ещё добавляется `[начало, конец, шаг]`

-- Следующий пример также показывает, что данный синтаксис может использоваться и в цикле `for`,
-- также как и другие генераторы.

for item in *points[1, 10, 2]
    print unpack item

-- Любые нежелательные значения могут быть отброшены. Вторая запятая
-- не требуется если не указан шаг.

words = {"these", "are", "some", "words"}
for word in *words[,3]
    print word

-----------------------------------------------------
-- 4. Управляющие структуры
-----------------------------------------------------

have_coins = false
if have_coins
    print "Got coins"
else
    print "No coins"

-- Используйте `then` для однострочного `if`
if have_coins then "Got coins" else "No coins"

-- `unless` это `if` наоборот
unless os.date("%A") == "Monday"
    print "It is not Monday!"

-- `if` и `unless` могут использоваться как выражения
is_tall = (name)-> if name == "Rob" then true else false
message = "I am #{if is_tall "Rob" then "very tall" else "not so tall"}"
print message -- "I am very tall"

-- `if`, `elseif`, и `unless` могут вычислять присвоение также как и выражения.
if x = possibly_nil! -- назначает `x` равным `possibly_nil()` и вычисляет `x`
    print x

-- Условия могут ставиться как после операторов, так и перед ними.
-- Это называется "декоратор строки".

is_monday = os.date("%A") == "Monday"
print("It IS Monday!") if isMonday
print("It is not Monday..") unless isMonday
--print("It IS Monday!" if isMonday) -- Это не оператор, не работает
```

```
--------------------------------------------------
-- 4.1 Циклы
--------------------------------------------------

for i = 1, 10
    print i

for i = 10, 1, -1 do print i -- Используйте `do` для однострочных циклов.

i = 0
while i < 10
    continue if i % 2 == 0 -- Оператор continue; пропускает оставшуюся часть цикла.
    print i

-- Циклы можно использовать в качестве декоратора строки, также как и условия
print "item: #{item}" for item in *items

-- При использовании циклов как выражений создается таблица-массив. Последний оператор
-- в блоке приведен к выражению и добавлен в таблицу.
my_numbers = for i = 1, 6 do i -- {1, 2, 3, 4, 5, 6}

-- используйте `continue` для отфильтровки значений
odds = for i in *my_numbers
    continue if i % 2 == 0 -- работает противоположно `when` в генераторах!
    i -- Добавлена только чтобы вернуть таблицу если нечетная

-- Цикл `for` возвращает `nil` когда он последний оператор в функции
-- Используйте явный `return` для создания таблицы.
print_squared = (t) -> for x in *t do x*x -- возвращает `nil`
squared = (t) -> return for x in *t do x*x -- возвращает новую таблицу квадратов

-- Указанное ниже делает то же, что и `(t) -> [i for i in *t when i % 2 == 0]`
-- Но генератор списков создаёт лучший и более читабельный код!

filter_odds = (t) ->
    return for x in *t
        if x % 2 == 0 then x else continue
evens = filter_odds(my_numbers) -- {2, 4, 6}

--------------------------------------------------
-- 4.2 Операторы выбора
--------------------------------------------------

-- Операторы выбора это сокращенный способ написания множества операторов `if`
-- проверяющих одно и то же значение. Значение оценивается единожды.

name = "Dan"

switch name
    when "Dave"
        print "You are Dave."
    when "Dan"
        print "You are not Dave, but Dan."
    else
        print "You are neither Dave nor Dan."

-- Операторы `switch` могут быть также использованы как выражения, а также при сравнении множества
-- значений. Значения должны быть на той же строке, что и `when`, если они в одном выражении.

b = 4
next_even = switch b
    when 1 then 2
    when 2, 3 then 4
    when 4, 5 then 6
    else error "I can't count that high! D:"
```

```
-----------------------------------------------------
-- 5. Объектно-ориентированное программирование
-----------------------------------------------------

-- Классы создаются при помощи ключевого слова `class` стоящего рядом с идентификатором,
-- обычно записанного КэмелКейсом. Значения, характерные для класса могут использовать @
-- как идентификатор вместо записи `self.value`.

class Inventory
    new: => @items = {}
    add_item: (name)=> -- обратите внимание на использование толстой стрелки для классов!
        @items[name] = 0 unless @items[name]
        @items[name] += 1

-- Функция `new` внутри класса является особенной, потому что она вызывается
-- когда создается экземпляр класса.

-- Создать экземпляр класса можно просто вызвав класс как функцию.
-- При вызове функций класса используется \ для отделения экземпляра от вызываемой функции.

inv = Inventory!
inv\add_item "t-shirt"
inv\add_item "pants"

-- Значения, определённые в классе - но не функция new() - будут общими
-- для всех экземпляров класса.

class Person
    clothes: {}
    give_item: (name)=>
        table.insert @clothes name

a = Person!
b = Person!

a\give_item "pants"
b\give_item "shirt"

-- выводит как "pants" так и "shirt"

print item for item in *a.clothes

-- Экземпляры класса имеют значение `.__class`, которое равно классу объекта,
-- чей экземпляр создан.

assert(b.__class == Person)

-- Переменные, объявленные в теле класса используя `=` являются локальными,
-- таким образом данные "частные" переменные доступны только в текущем контексте.

class SomeClass
    x = 0
    reveal: ->
        x += 1
        print x

a = SomeClass!
b = SomeClass!
print a.x -- nil
a.reveal! -- 1
b.reveal! -- 2

-----------------------------------------------------
-- 5.1 Наследование
-----------------------------------------------------
```

```
-- Ключевое слово `extends` используется для наследования свойств и методов
-- из другого класса.

class Backpack extends Inventory
    size: 10
    add_item: (name)=>
        error "backpack is full" if #@items > @size
        super name -- вызывает Inventory.add_item со значением `name`.

-- Т.к. метод `new` не был добавлен, вместо него будет использован
-- метод `new` из класса `Inventory`. Если мы хотим использовать конструктор с применением конструктора из
-- класса `Inventory`, мы должны использовать магическую функцию `super` при вызове `new()`.

-- Когда класс расширяет другой класс, он вызывает метод `__inherited`
-- в родительском классе (если он есть). Он всегда вызывается родительским и дочерним объектом.

class ParentClass
    @__inherited: (child)=>
        print "#{@__name} был наследован классом #{child.__name}"
    a_method: (a, b) => print a .. ' ' .. b

-- Будет напечатано 'ParentClass был наследован классом MyClass'

class MyClass extends ParentClass
    a_method: =>
        super "hello world", "from MyClass!"
        assert super == ParentClass

------------------------------------------------
-- 6. область видимости
------------------------------------------------

-- Все значения по-умолчанию локальны. Для объявления переменной как глобальной
-- используется ключевое слово `export`.

export var_1, var_2
var_1, var_3 = "hello", "world" -- var_3 локальная, var_1 - нет.

export this_is_global_assignment = "Hi!"

-- Для того, чтобы сделать классы глобальными также используется `export`.
-- Кроме того, все переменные в КэмелКейсе могут быть автоматически экспортированы при помощи
-- `export ^`, и все значения могут быть экспортированы указанием `export *`.

-- `do` позволяет вам вручную создать область видимости, в которой требуются локальные переменные.

do
    x = 5
print x -- nil

-- Здесь мы используем `do` как выражение для создания замыкания

counter = do
    i = 0
    ->
        i += 1
        return i

print counter!  -- 1
print counter!  -- 2

-- Ключевое слово `local` используется для определения переменных до их назначения

local var_4
if something
    var_4 = 1
```

```
print var_4 -- работает, потому что `var_4` была назначена в данной области видимости, не в области `if`

-- Ключевое слово `local` может быть использовано также для затенения существующей переменной

x = 10
if false
    local x
    x = 12
print x -- 10

-- Используйте `local *` для объявления всех идущих следом переменных.
-- Таким же образом, для объявления всех переменных в КэмелКейсе используйте `local ^`.

local *

first = ->
    second!

second = ->
    print data

data = {}

-----------------------------------------------------
-- 6.1 Import
-----------------------------------------------------

-- Значения из таблицы могут быть перенесены в текущую область видимости при помощи ключевых слов `import` и `from`.
-- Имена в списке импорта могут предваряться `\` если они являются модульной функцией.

import insert from table -- local insert = table.insert
import \add from state: 100, add: (value)=> @state + value
print add 22

-- Как и в таблицах, запятые можно исключить из списка импорта для обеспечения более длинных списков

import
    asdf, gh, jkl
    antidisestablishmentarianism
    from {}

-----------------------------------------------------
-- 6.2 With
-----------------------------------------------------

-- Оператор `with` можно использовать для быстрого вызова и присваивания значений
-- в экземпляре класса (объекте).

file = with File "lmsi15m.moon" -- `file` это значение `set_encoding()`.
    \set_encoding "utf8"

create_person = (name, relatives)->
    with Person!
        .name = name
        \add_relative relative for relative in *relatives
me = create_person "Ryan", {"sister", "sister", "brother", "dad", "mother"}

with str = "Hello" -- назначение как выражение! :D
    print "original: #{str}"
    print "upper: #{\upper!}"

-----------------------------------------------------
-- 6.3 Деструктуризация
-----------------------------------------------------

-- Деструктуризация может применяться с массивами, таблицами и вложенными таблицами для конвертирования их в локальны
```

```
e переменные

obj2 =
    numbers: {1, 2, 3, 4}
    properties:
        color: "green"
        height: 13.5

{numbers: {first, second}, properties: {:color}} = obj2

print first, second, color -- 1 2 green

-- `first` и `second` возвращают [1] и [2] потому что они обрабатываются как массив, но
-- `:color` принимается как `color: color`, таким образом он выставляет себя в значение `color`.

-- Деструктуризацию можно использовать вместо импорта.

{:max, :min, random: rand} = math -- переименовывает math.random в rand

-- Деструктуризацию можно выполнить везде, где может быть выполнено присвоение

for {left, right} in *{{"hello", "world"}, {"egg", "head"}}
    print left, right
```

Полезные библиотеки

Тут перечислены сторонние библиотеки, которые довольно минималистичные и полезные, чтобы их можно было использовать при написании игр под **TIC-80**.
Не всегда можно использовать стороннюю библиотеку ввиду ограниченного объёма памяти **TIC-80** выделенного под код - всего **64 кб**. Но заглянув в них, всегда можно научиться чему-то новому.

Что нужно для того чтобы самому подключить стороннюю библиотеку написанную на Lua?
Как правило, подключение выглядит так. Вместо использования `require имя_файла_библиотеки` нужно скопировать содержимое файла библиотеки, удалить последнюю строчку содержащую `return имя_переменной` и далее в своем коде игры обращаться к переменной `имя_переменной`.

Flux (Lua)

Быстрая, легкая библиотека для твининга, с функциями плавности и возможностью группировать твины вместе.

Запустить пример использования библиотеки

Lume (Lua)

Библиотека содержащие утилитные функции, помогающие в написании игр.

Запустить пример использования библиотеки

Bump (Lua)

Библиотека обнаружения коллизий. Ввиду своей минималистичности, скорости работы и простоты в использовании она прекрасно подходит для создания игр под **TIC-80**.

Запустить пример использования библиотеки

LZW image compression

Пакет для архивирования картинки LZW алгоритмом.
Сжатую картинку можно встроить в код в виде текстовой строки.
Проект на github

Запустить пример использования библиотеки

TICuare (Lua)

Простая и настраиваемая UI библиотека для TIC-80, основана на библиотеке Uare.
Проект на github

Запустить пример использования библиотеки

PSLIB (Lua)

Продвинутая библиотека системы частиц для TIC-80.

Запустить пример использования библиотеки

PICO-8 Wrapper for the TIC-80 (Lua)

Библиотека-обертка, для портирования игр с PICO-8 на TIC-80.

Проект на github

Вопросы и ответы

- **Как запустить скачанный картридж?**

Первый вариант: в консоли TIC-80 запускаем команду `add`, чтобы добавить его в папку с играми TIC. Затем его можно загружать при помощи команды `load` и запускать/редактировать. Вариант второй: запускаем команду `folder` - она открывает системную папку TIC, в неё при помощи проводника/файлового менеджера копируем картридж, далее грузим при помощи `load`.

- **Скопировал и вставил исходник, а он не запускается, пишет ошибки. В чем может быть дело?**

Скачайте и запускайте самый свежий TIC-80! Картридж с игрой это не только исходный код, но и такие данные как: графика, карта тайлов, звуки и музыка. Чтобы скачать весь картридж нужно: сохранить картридж в системную папку TIC-80 с помощью команды консоли `save имя_файла`, теперь его можно увидеть в системной папке с помощью `dir`, затем скачать его командой `get имя_файла.tic`. Удобнее всего набрав несколько символов имени файла, нажать TAB и консоль сама дополнит все оставшиеся символы.

- **Скопировал Lua-код из картриджа PICO-8, а он не работает в TIC-80, почему?**

Lua-код "фантазийной" консоли PICO-8 немного отличается от стандартного Lua-кода, также методы API PICO-8 отличаются от методов API TIC-80. Практически любой код PICO-8 необходимо портировать для TIC-80.

- **В TIC-80 подключен Lua 5.3 и есть такие математические функции как atan2, cosh, sinh, tanh, pow, frexp, ldexp. Я читал что в Lua 5.3 их удалили.**

В TIC-80 Язык Lua 5.3 собран в режиме совместимости с 5.2 (параметр `LUA_COMPAT_5_2`) Прежде всего это заметно в модуле math - такие функции как: atan2, cosh, sinh, tanh, pow, frexp, ldexp оставлены из версии 5.2, в то время как из обычной версии Lua 5.3 их исключили.

- **Можно ли редактировать код в отдельном редакторе и потом запускать в TIC-80?**

Да можно. Первый вариант: нужно разместить ваш файл game.lua рядом с исполняемым файлом tic. Во встроенном редакторе кода наберите dofile("game.lua") в первой строке. Теперь изменив код в стороннем текстовом редакторе Вы можете перезапустить картридж TIC, например нажав комбинацию CTRL+R Второй вариант: начиная с версии 0.21 можно запускать TIC с параметрами. Выполнив команду в консоли операционной системы `tic -code game.lua` Вы запустите TIC-80 в который будет встроен код из указанного Вами файла. Примечание: если вы используете MoonScript можно сохранять исходный код с расширением .moon, но и не забывайте указать тэг вначале кода — moon

Список изменений

version 0.47.0

In .47 we got help from 4 contributors:
Matheus Lessa Rodrigues
Anthony Camboni
Filippo
Death
Thanks!

All the editor modes use the only system palette now, except Sprite and Map Editors (they use game palette), added LUA live reload command line parameter (use -code-watch) and closed other small issues...

Closed issues

- On website, SURF is broken bug #303
- Use scanline trick to draw Sprite and Map editors with cart palette enhancement #345
- better bounds checking for memcpy, memset bug #342
- Live reload of .lua file(s) in developer mode enhancement #299
- Tic 0.47.0 enhancement #340
- btn() always returns number api bug #339
- Crash on Android android bug #324
- Ensure tile pos is always visible in map editor enhancement #338
- adding code editor ctrl+home and ctrl+end functionality editors enhancement #334
- Run button on code editor only editors enhancement #331
- textri() causes error when using javascript api bug #328

version 0.46.0

- Music Editor can copy/paste selected area now
- added on/off green buttons to temporarily mute the channel
- CTRL+scroll changes note value of selected notes

Also, we have first pull request from @Nullious! he added textri api function which you can use to draw a triangle filled with texture from image ram or map ram

Closed issues:

- Music Editor requests #174
- Add buttons to toggle channels on/off in music editor #193
- bug in SURF #304
- textri command api #317
- Browser hang if i run html-build, in native build OK. JavaScript. #306
- Border color does not update when loading carts until esc is pressed. #308
- V 0.45.0 exported doesn't contain correct palette #307

version 0.45.0

- In code editor, keep the current column of the cursor when switching lines #184

- Incorrect Lua syntax highlighting in text editor #232
- NEW with wrong parameters gives a silent error #257
- Add "resume" command to return to the game in the same place where the game was stopped #264
- Ability to edit default game template in similar way to config #276
- time() function returns different values depending on how the program is run #278
- error() crashes TIC #279
- Game menu doesn't work if game has mouse input #280
- Shortcut CTRL+/ in JavaScript code added LUA comment symbols "--" #284
- TIC website improvements #285
- Game menu wraps downwards but not upwards #286
- Implement SURF mode #293
- SURF return to TIC doesn't restore previous line index (and other suggestion) #296
- SURF doesn't show cover for local cart #295
- If you select "reset game" in Game Menu the sound will break - an annoying hiss will begin. #297

version 0.40.0

- Game menu #270
- Edit Config as cartridge #269
- Confirmation Dialogs #59
- GIF recording #157

If you press ESC (or BACK on mobile) you'll see Game Menu where you can resume/reset the game, configure gamepad buttons mapping or exit to the editors. config.lua is deprecated and it looks like cartridge now, in code you define variables, system images in sprites and systems sounds in SFX editor accordingly. As usual use config command to load configuration, config reset to reset. You will see WARNING dialog:

- if you have unsaved changes and try to load/new/exit commands
- if you try to delete a file using del command
- if cartridge already exists and you try to overwrite it
  Press F9 to start/stop GIF recording (you'll see REC label in the top-right corner)
  Also, you can define these variables in the Config:
  GIF_LENGTH=20 — in seconds
  GIF_SCALE=2
  And finally, we have the same ARM build PocketCHIP and RaspberryPi.
  Hope you enjoy it!

version 0.30.0

- Music editor: Row indicator #248
- Map Editor (0.30.0): Fill in selected box only (Suggestion) #249
- CLIP doesn't affect CLS api #247
- Show 'press BACK button to enter UI' on mobile #237
- Don't copy internal files to separate folder on every release #246
- Unflip the nibbles in clipboard data #226
- CTRL+Enter to play pattern from cursor pos in the Music Editor #238
- [REQUEST] Fill a portion of the map with a specific tile #138
- Box select, copy and paste in the map editor #132
- Map editor request #178
- Export native not working in 0.29.3 #233

- Random generator seed doesn't work on MacOS #228
- JS: sfx() incorrectly using "note" argument #229
- JS: font() produces incorrect results #230
- Array of transparent colors for SPR() #225

version 0.29.3

- Fixed performance issue in HTML export
- Added PocketCHIP build
- Array of transparent colors for SPR() api #225
- Fixed game crash exported to html
- JavaScript support #213
- Error message when run out of code space #149
- Replace color function in sprite editor editors #183
- Timeout version check after certain period passed #216
- Gridlines not visible in the map editor when the background sprite isn't black #190
- Change the startup sound #130
- Add 'saveid:' metatag #199
- Command line loaded sprites with wrong palette indices? #200
- Add ls command the same as dir #201
- Can't save or load a cart in Windows 8.1 #202
- Cannot change palette for new carts in config file #204
- More accurate timer #208
- Extra spaces on command line interpreted as part of file name #215
- Help commands show capital instead of lowercase #214
- wasted code when injecting it from the command line #219
- Crash in the Music Editor bug #218

The main enhancement is Javascript support, use new js command to load Hello World demo

version 0.28.1

- official 80K RAM layout you can access all the memory by peek/poke functions
- TIC has 60 music patterns in total which can be assigned to any channel and any track
- fixed bug with sfx api, you couldn't play effect on two channels at the same time
- add import/export map command to get the map as raw binary (240*136 bytes), also you can inject map from command line tic cart.tic -map world.map
- added X Y gamepad buttons handling (also you can configure binding in keymap)
- add gamepad mask to assign what buttons show and handle (00111100 by 0x03FFB address will show LRAB only)
- add frame and loop parameters to music api music [track=-1] [frame=-1] [loop=true]
- Loading cart and code via command line does not load the palette within the cart.

HOTFIX

- double ESC returns to game
- cursor doesn't render (only black box)
- X Y touch buttons #186 #185
- added NUMPAD support to enter numbers in the Music Editor
- enable record mode in music editor and press Enter, cursor moves to pattern edit box!!!

- fixed typo "ame is missing"
- fixed cover loaded from the previous cart

version 0.27.1

- Record option for the music editor enhancement #167
- Sprite editor enhancements enhancement #128
- Feature: Color picker and other keyboard shortcut in sprite editor. enhancement #124
- Serious "loss of data" issue with new command bug #168
- loadmusic/loadsfx/loadsprites/loadmap/loadcode commands enhancement #164
- [0.26.0 dev Windows 10 exe] font function crashes program when given nil value bug #165
- trace() is useless for truth/false values enhancement #163
- Volume column in tracker? enhancement #161
- Palette editor enhancement #166

Console command changes:

- added config reset to reset config.lua
- added load cart.tic [sprites | map | cover | code | sfx | music | palette] to load sections from other carts

API changes:

- trace output works like standard Lua print function
- changed RGB order by 0x3FF0 from BGR to RGB

HOTFIX

- Can't load code with -code argument on 0.27.0 #172
- wrong pallete when import sprites from gif. #171

version 0.26.0

- Restore sprites and map data on every game start #159
- Update check or autoupdater #153
- If custom cursor is defined in config.lua, it still doesn't allign with the resolution #155
- Screenshots with Hotkey #144
- You can cd into non-existing directories online #148
- TIC doesn't load demo carts on win7 #142
- Cover image palette changed #93

version 0.25.0

- Sfx editor improvements #140
- Soft Keyboard does not reappear after running game with mouse input. #66
- Command line argument to inject an image into the launched cart #141
- Version mobile on scrolling #120
- Change screen offset in every scanline #129
- Browse and load carts directly from the tic.computer website #146
- 0.24 crash with moonscript error #135
- Feature: Custom colors defined in metadata for sprites and palette in sprite editor #119
- Pre-seed the random generator in the Lua environment #127

version 0.24.0

- On V 0.23 Demo SOUND should be named SFX #118
- SFX editor keybaord keys not working correctly if NumLock is active #114
- Feature: Colored output of trace() #121
- Sprite editor: copy > scale > paste bug #115
- Scaling the map #75
- Mouse cursor #116
- Border scanline effect #126
- Add "cd" command for the console #82
- Configure buttons mapping #45

version 0.23.0

- sound module has been rewritten from scratch and got editable waveforms
- added Music category to the https://tic.computer/play?cat=3
- added speed parameter to the tracker (for fine tune music speed, means how many 'ticks' to play each row)
- fixed arpeggio note pitch
- Map same keys for SFX and MUSIC Editor #112
- Add the ability to output log to file. #109
- btnp() hold parameter time resets when any other button is pressed #108
- Cannot print newlines #90
- Improvement the "world map" in the "map editor" #79
- Show the tile's ID for the map editor. #105
- Scaling the text with print() function #78
- Cannot load files that includes whitespace in its filename #88
- "sfx(-1)" is required before playing same sound, duration parameter to sfx api. #87
- add hotkey for comment/uncomment #95
- Sprite editor - Selection tool bug #102
- feature: sspr() function: multi tile sprite operations and palette swapping #71

API changes:

- you can draw composite sprites, add w and h parameters
  spr id x y [colorkey=-1] [scale=1] [flip=0] [rotate=0] [w=1 h=1]
- added duration parameter to sfx, means how many ticks to play (sfx == -1 by default to play sfx infinitely)
  sfx id [note] [duration=-1] [channel=0] [volume=15] [speed]
  where note is index = octave * 12 + note or string like C#4
- added scale param to print and font api
  print text [x=0 y=0] [color=15] [fixed=false] [scale=1] -> width
  font text [x=0 y=0] [color=15] [w=8 h=8] [fixed=false] [scale=1] -> width
  where w and h size for empty symbol (space for example)
  also print and font can draw multiline text (use \n to wrap words)

version 0.22.0

- added first version of the Music Editor
- redesigned SFX module, you can call sfx just by id (unfortunately you have to fix sfx in old carts)
- added MUSIC api function

- added CLIP api function
- added TRIM api function

version 0.21.0

- you can assign cart cover image by pressing [F7]
- added 'import/export cover' command
- cover loads to screen buffer on every game start
- [github] EXPORT behaves differently on Windows and HTML platforms #64
- [github] Native Scrolling on macOS #67
- [github] Cursor Jump #68
- [github] dofile on first line false detection #65
- [github] Running the MoonScript code using 'dofile' produces an error #70
- [github] Add the ability to run the tic application from the command line with the following parameters: -cart game.tic, -code code.lua #69

version 0.20.0

- [github] Configuration for startup sound #51
- [github] Sound RAM Poke Crash #55
- [github] Touch buttons with labels #56
- [github] Triangle primitives #63
- added toolbar icons with tooltips
- added THEME to the config (you can define start beep, code syntax colors, touch gamepad buttons)
- copy/paste works in SFX EDITOR
- reduced Linux builds size from 8MB to 2MB
- added TRI api function to draw filled triangles

version 0.19.0

- [github] sprites flag #40
- [github] Syntax coloring for moonscript #47
- [github] Second gamepad not processed correctly. #43
- [github] Error occurs when i try to import GIF, OS Windows 7 #50
- [github] Add copy/cut/paste buttons to code and sprite editor #49
- [github] Circ not draw correctly with x < 0 #53
- [github] A bug in Android version #4
- [github] Export/Import on Android #54
- [github] Moonscript: dynamic code preview does not display the names of methods #32
- fixed gamepad B button in UWP version
- win phone don't work with ctrl+c/v, added ctrl/shift+insert

version 0.18.0

- [github] Cartridge metadata editor #37
- [github] Mouse support #36
- [github] Add Palette in Metadata Cartdrige #38
- [github] TIC's r-w directory requires root to edit and view on Android. #39
- [github] Other Api method #41

- [github] Console commands for export/import spritesheets. #20
- [github] Add new console command 'folder' #30
- added GIFLIB support
- added number to name: TIC-80

version 0.17.0

- [github] DIR console command #24
- [github] The moonscript does not run if you run it with the command dofile('game.lua') #25
- [github] Add standard LUA library 'coroutine' #26
- [github] set back 'demo' command on 0.16.0 #22
- [github] font customization for all characters #15
- [github] Can't save clean cart #34
- [github] shake screen API function #33
- added FONT api func to render variety width font
- you can set screen border color by 0x3FF8 in the RAM
- added Windows XP support

version 0.16.0

- [github] mouse scroll #9
- [github] [Web export] Sound lags #13
- [github] Hotkeys for Mac (again :-) #10
- text renders with variety width by default (added 'fixed' param to 'print' api func)
- default font and palette loads from config.lua
- added wide font version (made by Fred Bednarski @level27geek) and DB16/PICO8/ARNE16/EDG16/A64/C64 palettes to config.lua

version 0.15.0

- [github] Moonscript support #8
- [github] HOTKEYS rectification #5
- [github] Multiline comments not rendered correctly in code editor #6
- [github] No support for OS X 10.11 (El Capitan) read about -mmacosx-version-min=version (10.7) flag
- scanline trick works only in RUN mode
- new export command format: export [html | native]

version 0.14.0

- added Mac OS X support
- [github] TIC crash when reach 65536 char limit #1
- [github] pbtn clarification #2 btnp [id [hold period]] -> pressed, it will return true every 'period' ticks if button is held for 'hold' ticks
- added crossplatform experimantal tool 'TIC bundler' to build HTML and native (win/linux/macos) games
- added 'demo' command to the console
- fixed some crashes

version 0.0.13

- reduced app exe size on Windows (from 6.1MB to 1.8MB)
- sound registers depend of frequency instead period (here you can read how to calculate frequency for note https://en.wikipedia.org/wiki/Piano_key_frequencies) you can change audio channel wave form in runtime (see how to do it in demo_sound.tic)
- new 64K RAM layout (use 'ram' command in the console to learn more)

  64K RAM layout

  - **0000**-SCREEN
  - **3FC0**-PALETTE
  - **3FF0**-PALETTE MAP
  - **3FF8**-(reserved for future use)
  - **4000**-SPRITES
  - **8000**-MAP
  - **FF80**-SOUND REGISTERS
  - **FF90**-PERSISTENT MEMORY
  - **FFD0**-(reserved for future use)
- persistent memory resized to 16 integers (64 bytes according to the new 64K RAM)
- removed 'sndreg' api function, you have to set/get sound registers values by 0xFF80 addr in the RAM
- added 'time' api function (returns how many tics have passed since the game started)
- create issue tracker on GitHub (https://github.com/nesbox/tic.computer)
- removed all compilation warnings
- fixed some crashes

version 0.0.12

- added 'sndreg' api function to have low level access to the sound synthesis: TIC has 4 sound channels each with 3 registers (period/volume/timbre) and you can dinamically generate any music or sfx in runtime (I'm getting ready to implement music tracker...)
- added 'btnp' api function, returns true if button is pressed and wasn't pressed in previous frame
- added 'exit' api function
- added demo 'demo_palette.tic' where you can change palette (DB16, pico8, A64, C64, arne16, edg16)
- added demo 'demo_sound.tic' to demostrate how sound registers work

version 0.0.11

- export native Linux build (use 'export native' command)
- added flip/rotate/del buttons to the Sprite Editor
- open .tic files directly from OS
- tab autocomplete in the Console
- scroll in the Console
- added CLS command to clear Console screen
- 'export' works without saving on local FS (usage: export [html native])
- middle-click to select color in Sprite Editor
- fixed Code Editor history desync
- fixed Console text shifting in Linux
- fixed Linux window icon

version 0.0.10

- added persistent memory API (use pmem func to load/save 64 int values)
- added Select, Move and Bucket to Sprite Editor
- added canvas resizing to Sprite Editor
- added optional params [volume] and [speed] to sfx api
- 'pix' API func returns color
- middle click to select sprite in Map Editor
- fixed search direction in Code Editor
- fixed lua log10 bug on uwp and android
- created public collection with TIC games on nesbox.itch.io

version 0.0.9

- added CTRL+O to show Code Outline
- added dynamic preview to Find and Goto Line in code editor
- disabled gyroscope in android by default

version 0.0.8

- added rotate param to the SPR function values [0,1,2,3] means 0,90,180,270 degrees
- added CTRL+F shortcut to find text in the code editor
- added CTRL+G shortcut to jump to line in the code editor with poke4/peek4 you can set/get half-byte values in RAM
- added sprite palette remaping, use peek4/poke4 by address 0x3FF0
- drag source code with right mouse button
- app window scaling is divisible by integers
- loaded cart name is in windows title
- SFX autostops when ID or NOTE is changing
- added simple SFX demo

version 0.0.7

- added Android build
- added Linux x64 build
- added joystick/gamepad support (only one player so far)
- added foreground/background sprites
- added world map preview
- right click to select second color in sprite editor
- added demo_p3d.tic demo created by Filippo Rivato (@HomineLudens)

version 0.0.6

- added undo/redo changes to all the editors (CTRL+Z/CTRL+Y)
- fixed screen keyboard input and rendering in UWP
- fixed SFX command, just call once sfx(id) to play, sfx() to stop
- added mouse scrolling in the code editor
- added TAB/SHIFT+TAB to format selected code block
- added 'trace(msg)' API command to print message in th console

- added mapping for the second player: RFDG, A,S
- added two demo projects, demo_ttt.tic (TicTacToe) and demo_fire.tic created by Filippo Rivato (@HomineLudens)

version 0.0.5

- added CRTL+S to save project
- copy/paste in sprite editor, DEL to delete sprite
- spritesheet navigation by arrows in sprite editor
- fixed texture rendering in UWP
- RAM command shows memory addresses in console
- added memcpy,memset API functions
- added mget,mset API functions to get/set map tile
- map moving by arrows buttons
- added pin button to spritesheet in map editor
- added console history
- added NEW command to create empty project

version 0.0.4

- changed display resolution in spec 240x136
- added new font with 6x6 symbols with lowercase support
- new UI layout for SPRITE and MAP editors
- added grid and current pos to the MAP editor
- added fullscreen mode (F11, ALT+ENTER)
- added transparent color for sprites rendering
- updated to SDL 2.0.5
- CTRL+X works in code editor
- BACK buttons works on mobile (like escape)
- fixed some crashes

version 0.0.3

- fixed CPU overhead when app is minimized
- new project name - TIC, .tic extension for cartridges and http://tic.computer domain for project
- 'tic' game loop function
- scanline function support, you can change the palette per render line (102 time per tick) and show 102*16=1632 different colors per tick
- ADD command works in HTML version
- EXPORT command makes standalone html/exe game (works only in native versions)
- UWP build for Windows Store

Трекер задач

Не стесняйтесь открыть новую задачу если у Вас ошибка или Вам нужна новая функция.
Вы можете просмотреть существующие задачи или создать новую.

Официальный язык - английский. Спасибо!

Благодарности

Filippo Rivato -Twitter @HomineLudens
Fred Bednarski -Twitter @FredBednarski
Trevor Martin - Twitter @trelemar

Составление русскоязычной версии документации: Al Rado -Twitter @alrado2
Помощь в переводе статьи "Изучаем Lua за 15 минут": Никита Курылев Twitter @nikita_kurilev

Редактирование документации и перевод статьи "Изучаем MoonScript за 15 минут": Михаил Радюк -Twitter @torabora08