

# Графика в PicoMite VGA

## Введение

PicoMite — это автономный компьютер, на котором работает MMBasic. Подробнее можно прочитать тут:

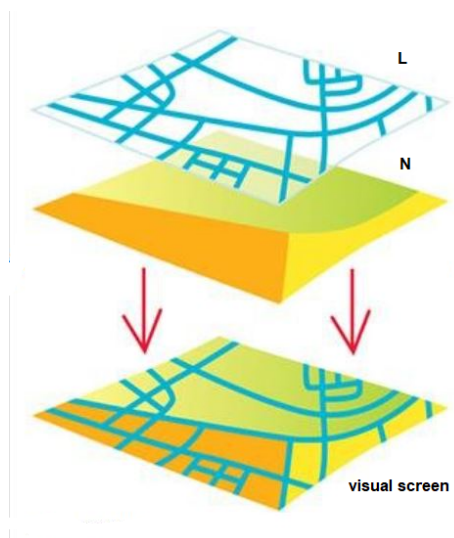
<https://geoffg.net/picomitevga.html>

Базой для этих компьютеров является Raspberry Pi Pico — небольшой модуль на базе чипа RP2040.

В этой статье делается попытка объяснить, как можно использовать графику в MMBasic в PicoMite. Большая часть этого подходит для разных версий PicoMite, но по сути основана на PicoMite VGA. Здесь будем ссылаться на игру PETSCII ROBOTS, использующую представленный метод.

## Слой графики

Для пользователя видеоэкран отображается как 2D-объект, плоская поверхность. Это называется слоем. Для программиста видеоэкран — это часть памяти. Это называется кадровым буфером (frame buffer). Эти термины используются в тексте ниже и относятся к одному и тому же.



## Слой (Layer) N

PicoMite запускается, используя только 1 графический слой (слой (Layer) N или кадровый буфер (framebuffer) N). В версии VGA это видимый экран, а при использовании ЖК-экрана — ЖК-экран. Если вы рисуете линию или печатаете текст на экране, он находится на слое N. Важно понимать, что разница между дисплеями VGA и LCD заключается в том, что слой N в ЖК-дисплее находится ВНУТРИ самого ЖК-дисплея, в его родном цвете, глубина. Для ILI9341 это 16-битный цвет (RGB5:6:5). Версия VGA хранит информацию внутри памяти чипа RP2040, и это разрешение 4 бита (RGB1:2:1).

## Кадровый буфер (FRAME BUFFER) F

При частом выводе на экран, возможно, вы увидите артефакты на экране. Чтобы избежать этих артефактов, вы можете выполнять всю запись в промежуточный буфер (FRAMEBUFFER F), а когда вы будете готовы выполнить все обновления графики, вы можете скопировать кадровый буфер F на слой N (FRAMEBUFFER COPY F,N). Обратите внимание, что кадровый буфер F находится внутри чипа RP2040 и имеет формат RGB 1:2:1. Если вы используете ЖК-дисплей, PicoMite автоматически преобразует RGB1:2:1 в RGB 5:6:5. Фреймбуфер F использует 38 КБ ОЗУ. Фреймбуфер F никогда не виден напрямую. Это просто блок памяти, который может содержать данные RGB 1:2:1.

## Слой (Layer) L/Кадровый буфер (Frame Buffer) L

Особенно полезно иметь несколько графических слоев, особенно для игр. PicoMite предлагает один слой L поверх (перекрывающий) слой N, называемый кадровым буфером L. Когда вы используете слой L, он визуально находится перед слоем N и может загромождать обзор слоя N. По этой причине слой L показан с «прозрачным цветом». Если вы нарисуете слой L этим прозрачным цветом, вы увидите графику слоя N. Если вы используете любой другой цвет, вы увидите пиксели слоя L. Прозрачным цветом по умолчанию является RGB (ЧЕРНЫЙ), но его можно изменить на любой из 16 цветов, которые поддерживает PicoMite. Часто используется RGB (пурпурный). FRAMEBUFFER L физически находится внутри чипа RP2040 и имеет размер 38 Кбайт. В версии VGA объединение N и L происходит автоматически. Использование кадрового буфера F не является обязательным. В версии с ЖК-дисплеем (помните, что L и N находятся в разных микросхемах) слияние между N и L невозможно. Версия с ЖК-дисплеем выполняет слияние в кадровых буферах F и L, а затем записывает результат на физический дисплей N.

Кадровый буфер N доступен по умолчанию. Другие фреймбуферы можно включить с помощью:

```
FRAMEBUFFER LAYER (для L)  
FRAMEBUFFER CREATE (для F) .
```

Для версии PicoMite VGA использование кадровых буферов полезно только в графическом РЕЖИМЕ 2 (320x240, 16 цветов).

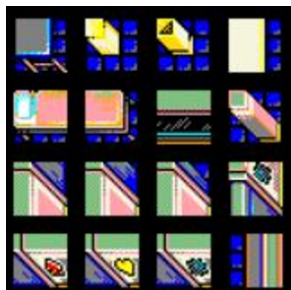
## Элементы графики

По сути, в PicoMite можно использовать два различных графических формата: плитки и спрайты.

Я не графический дизайнер, это мое название форматов.

Плитка — это прямоугольный графический объект, находящийся в собственном цветовом пространстве (т. е. RGB 1:2:1). Его можно скопировать на любой слой, и он виден напрямую.

Плитка может быть маленькой или размером с экран. PicoMite может загрузить плитку с устройства хранения (SD-карты) с помощью команды **LOAD IMAGE «имя\_файла.bmp»** и отобразить ее на активном слое. В качестве бонуса LOAD IMAGE также преобразует 24-битный RGB в RGB1:2:1 на лету. Файл должен быть в формате BMP.



Несколько тайлов от роботов PETSCII

Спрайт (sprite) — это прямоугольный графический объект, состоящий из индексов цвета (индексов). Каждый пиксель представлен не значением RGB, а индексом палитры. Это может сбивать с толку PicoMite, поскольку он поддерживает 16 цветов (RGB 1:2:1), но также поддерживает 16 индексов цвета в формате файла спрайта. И это НЕ одно и то же из-за обратной совместимости с CMM1, CMM2 (Colour Maximite <https://www.geoffg.net/maximite.html> ).

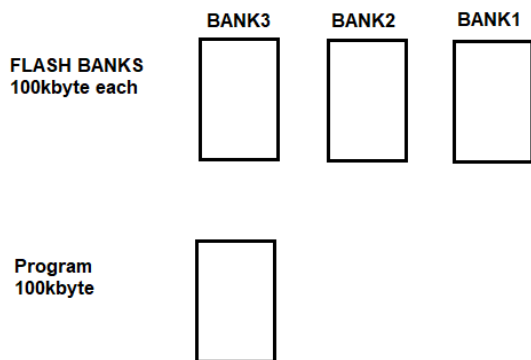
Данные спрайта в файле нельзя скопировать непосредственно на экран. Его необходимо преобразовать в значения RGB. Данные спрайта в файле состоят из заголовка, указывающего размеры спрайта (т. е. 16x8 пикселей), а затем списка (т. е. 128 (16x8)) индексов цвета. Спрайты — мощные элементы, они могут восстанавливать фон при перемещении, при движении есть обнаружение столкновений. PicoMite теперь поддерживает 64 спрайта (увеличено с 31 в бета-версии 5.09.00b0). Спрайты должны быть загружены в память с помощью **SPRITE LOAD «имя файла», номер спрайта**. После этого спрайт можно отобразить с помощью **SPRITE WRITE sprite\_number,x,y** (жесткая запись) или **SPRITE SHOW sprite\_number,x,y** (запоминает фон) и **SPRITE MOVE**.



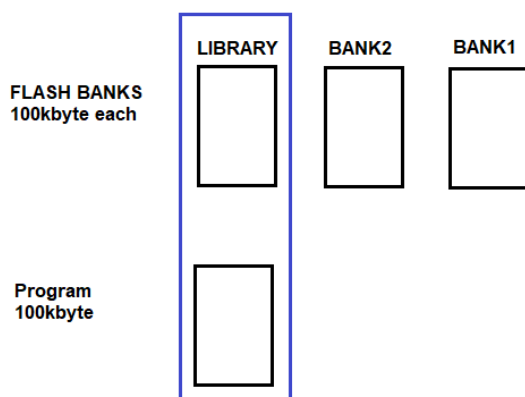
Некоторые спрайты с пурпурным фоном от роботов PETSCII.

## Память

RP2040 имеет ограниченный объем оперативной памяти, которая должна быть разделена между системой, видео, переменными, стеком, буферами (т. е. аудио) и т. д. Поэтому существуют возможности также использовать флэш-память для программ MMBasic.



Флэш-банки можно использовать для хранения программ MMBasic и даже запускать их. Вы можете СВЯЗАТЬ программы в банках. Подробности выходят за рамки данного документа. Просто знайте, что существует такая возможность. MMBasic также поддерживает БИБЛИОТЕКУ ( LIBRARY). Тот же самый блок флэш-памяти, который зарезервирован для BANK3, может использоваться как расширение программной памяти с ограничениями.



Ограничением является то, что вы не можете редактировать программу в библиотеке. Вы можете сохранить **CSUB**, **SUB** и **FUNCTIONS** в библиотеке, которая «без ошибок», и использовать их из основной программы. Поскольку библиотека активна во время работы программы MMBasic, доступ к ней можно получить только из командной строки. Если у вас в памяти программы есть отложенный код, перенесите его в библиотеку с помощью **LIBRARY SAVE**. **LIBRARY LIST** показывает, что находится в библиотеке, а **LIBRARY DELETE** стирает библиотеку и возвращает флэш-память во флэш-банк3.

## CSUB

Некоторые из вас, возможно, помнят, что в 80-х годах существовали базовые программы с множеством операторов DATA в конце, никто не знал, что они означают, но они засовывались в память, а затем выполнялись (CALL xxx). По сути, это был машинный код, представленный в виде шестнадцатеричных чисел. CSUB – это именно это. CSUB содержит инструкции машинного кода для процессора ARM в шестнадцатеричной форме. MMBasic немного более продвинутый, в нем не нужно запоминать куда POKE, а что

CALL. POKE выполняется автоматически при запуске программы, и просто используя имя CSUB, выполняется CALL xxx

Вывод: во время выполнения любой шестнадцатеричный код CSUB записывается в память в двоичной форме, чтобы подготовить его к выполнению ARM. Мы используем эту функцию, но не выполняем двоичный файл, а используем его для копирования.

Аналогично, когда вы сохраняете в библиотеке программу, содержащую CSUB, в библиотеку записываются двоичные данные (а не шестнадцатеричное текстовое представление).

## Реализация графики: подготовка

Теперь у нас есть знания, позволяющие понять графику, используемую в продвинутых играх, таких как РОБОТЫ PETSCII. PETSCII ROBOTS (для MMBasic) — большая игра. Базовая программа весит всего 70 кбайт, но она использует данные (карты, графику, звуки) размером более 800 кбайт. В игре используется более 100 спрайтов и около 300 тайлов. Учитывая ограничение максимум в 31 спрайт, при написании игры вышеуказанные знания использовались для реализации игры. Ниже приведены решения по игровому дизайну, которые были приняты для того, чтобы игра вписывалась в PicoMite.

1/ Используйте только кадровый буфер L в версии VGA, чтобы сэкономить ОЗУ для хранения переменных (карты/атрибуты). Недостатком этого метода является то, что при обновлении экрана вы можете увидеть незначительные искажения видео. В программе предусмотрены возможности минимизировать это за счет синхронизации тяжелых обновлений экрана с вертикальной синхронизацией с использованием **FRAMEBUFFER WAIT**.

Чтобы сделать это эффективным, важно выполнять весь код, записывающий в видеопамять, близко друг к другу. Не засоряйте весь код записью в видеопамять. PETSCII использует 2 **SUB**, выполняющихся в каждом игровом цикле. Один записывает все на слой N, а второй записывает все на слой L. И эти **SUB**'s настроены на скорость.

2/ Сохраняйте все графические элементы (тайлы/спрайты) в двоичном формате. MMBasic имеет высокооптимизированные процедуры копирования памяти из флэш-памяти в видеоОЗУ в **BLIT MEMORY** (или **SPRITE MEMORY**). Функция **BLIT MEMORY** работает быстро, но не может восстановить фон или обнаружить столкновения спрайтов.

Обнаружение столкновений выполняется в MMBasic на основе координат X и Y тайлов и спрайтов.

Восстановления фона можно избежать, используя 2 слоя. Слой N для мира (карта мира), построенный с использованием тайлов. Используйте слой L с пурпурным цветом в качестве прозрачного, чтобы написать спрайты. Когда они перемещаются, слой N не затрагивается.

## Храните спрайты и тайлы в двоичном формате.

При запуске **CSUB** или сохранении его в библиотеке шестнадцатеричные данные преобразуются в двоичную форму. Матерп написал базовую программу, которая преобразует файлы **SPRITE** (содержащие заголовок и индексы цвета) в заголовок + шестнадцатеричные данные RGB 1:2:1, имитируемые как **CSUB**.

'программа для преобразования всех файлов спрайтов в каталоге в CSUB с использованием сжатия, где это необходимо

Option explicit

Option default none

Const separatesubs% = 0

Dim offset%

Dim fname\$=Dir\$("\*.\*spr",FILE)

Open "tile0\_csub.bas" For output As #2

Open "tile0\_index.txt" For output As #3

If separatesubs%=0 Then

Print #2,"CSUB TILE0"

Print #2,"00000000"

offset%=0

EndIf

Do

If fname\$<>"" Then code fname\$

fname\$=Dir\$()

Loop Until fname\$=""

If separatesubs%=0 Then Print #2,"END CSUB"

Close #2

Close #3

'преобразовать файл f\$ в сжатый CSUB

Sub code f\$

Local i%,j%,h%,l%,w%,n%,s%,il%

Local a\$,o\$,oc\$

Open f\$ For input As #1

Line Input #1,a\$ 'process the dimensions and count

w%=Val(Field\$(a\$,1,""))

n%=Val(Field\$(a\$,2,""))

h%=Val(Field\$(a\$,3,""))

If h%=0 Then h%=w%

i%=Instr(f\$,".")

o\$=Left\$(f\$,i%-1)

If separatesubs%=1 Then

Print #2,"CSUB "+o\$

Print #2,"00000000"

offset%=0

' Else

' Print #2,""+o\$

EndIf

Local obuff%(w%\*h%\8+128),buff%(w%\*h%\8+128)

For s%=1 To n% 'process all the sprites in a file

Print #3,Str\$(offset%)

Print #2,"'Offset ";offset%

For l%=1 To h%

a\$=""

Do While Left\$(a\$,1)="" 'skip comments

Line Input #1,a\$

Loop

'убеждаемся, что все линии имеют правильную длину

If Len(a\$)<w% Then Inc a\$,Space\$(w%-Len(a\$))

If Len(a\$)>w% Then a\$=Left\$(a\$,w%)

LongString append buff%(),a\$ 'get all the file into a single longstring

Next l%

j%=0

```

For i%=1 To LLen(buff%())
    LongString append obuff%(),mycol$(LGetStr$(buff%(),i%,1))
Next i%
LongString clear buff%()
il%=(LLen(obuff%()+7)\8 * 8
i%=0
Do While i%<w%*h% 'compress the data
    j%=LGetByte(obuff%(),i%)
    l%=1
    Inc i%
    Do While LGetByte(obuff%(),i%)=j% And l%<15
        Inc l%
        Inc i%
    Loop
    LongString append buff%(), Hex$(l%)+Chr$(j%)
Loop
'вывод должен быть кратен 8 полубайтам
LongString append buff%(),Left$("00000000",8-(LLen(buff%()) Mod 8))
If LLen(buff%())<il% Then 'compressed version is smaller so use it
    Print #2,""+o$;
    Print #2," is compressed"
    Print #2,Hex$(h%+&H8000,4)+Hex$(w%,4)
    j%=0
    For i%=8 To LLen(buff%()) Step 8 'reverse the order
        o$=LGetStr$(buff%(),i%,1)
        Inc o$,LGetStr$(buff%(),i%-1,1)
        Inc o$,LGetStr$(buff%(),i%-2,1)
        Inc o$,LGetStr$(buff%(),i%-3,1)
        Inc o$,LGetStr$(buff%(),i%-4,1)
        Inc o$,LGetStr$(buff%(),i%-5,1)
        Inc o$,LGetStr$(buff%(),i%-6,1)
        Inc o$,LGetStr$(buff%(),i%-7,1)
        Inc j%
        If j%=8 Then
            Print #2,o$
            j%=0
        Else
            Print #2,o$+" ";
        EndIf
    Next i%
    If j%<>0 Then Print #2,""
    Inc offset%,4+LLen(buff%())\2
Else
    Print #2,""+o$;
    Print #2," is uncompressed"
    Print #2,Hex$(h%,4)+Hex$(w%,4)
    LongString append obuff%(),Left$("00000000",8-(LLen(obuff%()) Mod 8))
    j%=0
    For i%=8 To LLen(obuff%()) Step 8 'reverse the order
        o$=LGetStr$(obuff%(),i%,1)
        Inc o$,LGetStr$(obuff%(),i%-1,1)
        Inc o$,LGetStr$(obuff%(),i%-2,1)
        Inc o$,LGetStr$(obuff%(),i%-3,1)
        Inc o$,LGetStr$(obuff%(),i%-4,1)
        Inc o$,LGetStr$(obuff%(),i%-5,1)
        Inc o$,LGetStr$(obuff%(),i%-6,1)
        Inc o$,LGetStr$(obuff%(),i%-7,1)
        Inc j%
        If j%=8 Then

```

```

        Print #2,o$
        j%=0
    Else
        Print #2,o$+" ";
    EndIf
Next i%
If j%<>0 Then Print #2,""
Inc offset%,4+LLen(obuff%())\2
EndIf
LongString clear obuff%()
LongString clear buff%()
Next s%
Close #1
If separatesubs%=1 Then Print #2,"END CSUB"
End Sub
'
'преобразует цвет Ascii из стандарта Maximite в стандарт PicoMite
Function mycol$(c$)
Static cols%(15)=(0,1,6,7,8,9,14,15,2,3,4,5,10,11,12,13)
Local i%
If c$=" " Then c$="0"
i%=Val("&H"+c$)
mycol$=Hex$(cols%(i%))
End Function

```

Эта программа преобразует все спрайты в «filename.spr» в один файл «xxxx\_csub.bas» и файл «xxxx\_index.txt», который после преобразования в двоичную форму содержит все указатели на элементы в CSUB. Относительные позиции привязаны к начальному адресу CSUB. Имя CSUB (т. е. CSUB TILE0) важно для этого, поскольку CSUB будет много. Обратите внимание, что файлы в каталоге обрабатываются в порядке возрастания. Поэтому лучше всего нумеровать спрайты в именах файлов, чтобы порядок в CSUB был предсказуемым.

Затем «LOAD xxxx\_csub.bas», чтобы загрузить файл в память программы, и «LIBRARY SAVE», чтобы записать его как двоичные данные во флэш-память в библиотеке. Теперь у нас есть двоичные данные RGB 1:2:1 во флэш-памяти в неизвестном месте. Вы можете сохранить в библиотеке несколько файлов xxxx\_csub.bas, пока библиотека не заполнится. Чтобы найти абсолютный адрес каждого CSUB в памяти, вы извлекаете начало CSUB.

```
Address% = PEEK(CFUNADDR csub_name) ' то есть PEEK(CFUNADDR TILE0).
```

Для функции **BLIT MEMORY** лучше всего построить массив с абсолютными адресами. В приведенном ниже примере создается абсолютный индекс для 6 (0...5) спрайтов в CSUB под названием HEALTH.

```

'получаем начальные адреса
Local hlt=Peek(cfunaddr HEALTH)

'создаём файл глобального индекса
Dim health_index(5)

```



```

Open "sprites/hlt_index.txt" For input As #1
For i=0 To 5
    Input #1,a$

    health_index(i)=hlt+Val(a$)
Next
Close #1

```

Теперь вы можете использовать **BLIT MEMORY health\_index(3),X,Y** чтобы скопировать третий спрайт в (X,Y) на активном слое, заданном **FRAMEBUFFER WRITE L/N**.

## Из БИБЛИОТЕКИ во ФЛЕШ-БАНК (From LIBRARY to FLASH BANK)

Выше вы используете БИБЛИОТЕКУ для двоичных данных. Это требует подготовки библиотеки вручную с помощью командной строки, что не удобно для игры. Однако помните, что LIBRARY и FLASH BANK3 используют одни и те же места во Flash, но работают по-разному.

Программы в БИБЛИОТЕКЕ могут быть частью вашей активной программы, поэтому CSUB имеют значение в основной программе. FLASH BANK3 не является частью активной программы, поэтому CSUB не имеет значения.

Но начальный адрес банка флэш-памяти 3 (который может меняться в зависимости от версии MMBasic) можно запросить с помощью **flash\_address% = MM.INFO(FLASH ADDRESS 3)**. Это начальный адрес памяти для флэш-банка 3, а также начальный адрес памяти для библиотеки.

Вместо использования индекса на основе CSUB адаптируйте его так, чтобы он был индексом, ссылающимся на начало флэш-банка. Добавьте смещение к каждому индексу (разницу между «флэш-адресом» и «csb-адресом для этого csub») и создайте новый индексный файл, который ссылается на начальный адрес во флэш-памяти. Сделайте это для всех CSUB.

Теперь у вас есть индексный файл «flash\_index.txt», который ссылается на каждый графический элемент (спрайт/тайл) на начальный адрес FLASH BANK.

Затем сохраните библиотеку на SD-карту в виде двоичного файла.

```
LIBRARY DISK SAVE "filename.bin"
```

И сохраните эту библиотеку вместе с индексным файлом. Оба бесполезны друг без друга. Теперь мы можем очистить библиотеку и использовать флэш-пространство для FLASH BANK 3.

## Реализация графики: ИГРА

В игровой программе MMBasic мы теперь можем загружать двоичный файл во флэш-банк 1, 2 или 3, а все графические элементы можно найти, добавив индекс к стартовому адресу флэш-памяти того конкретного банка, который вы решите использовать в игре. Пример:

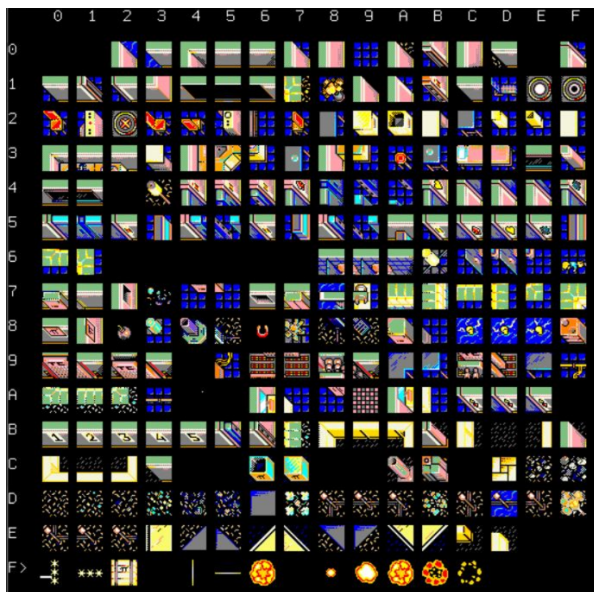
```
FLASH DISK LOAD 2, filename.bin,0      ' загружает двоичный файл во флэш-банк 2
Flash_address%=MM.INFO(FLASH ADDRESS 2) ' дает начальный адрес банка 2
```

После этого создайте массив **abs\_index%()** с абсолютными индексами, добавив **flash\_address %** ко всем относительным индексам, считанным из индексного файла «**flash\_index.txt**». Тогда просто: **BLIT MEMORY abs\_index(i),X,Y**

## ВАЖНО

### 1/ Документация

Важно сделать схему какой спрайт/тайл принадлежит какому индексу. При разработке PETSCII ROBOTS было полезно вести такую схему (она показывает номер плитки в массиве).



### 2/ СПРАЙТЫ

Вам нужны все графические элементы в виде файлов спрайтов (индексированные цвета и заголовков размера). Это можно реализовать несколькими способами, но для PETSCII ROBOTS используется способ загрузки файла BMP на экран, а затем попиксельный расчет индекса цвета. Затем записываем файл обратно на диск в виде файла спрайта. По сути, эту функцию выполняет следующая программа, написанная Martin\_H. Он пакетно обрабатывает 86 спрайтов размером 24x24.

```
'-----  
'Здесь информация об источнике  
FN$="spritesMix_bearbeitet.bmp"  
W=24  
H=24  
num=86  
'-----  
,  
  
Dim Col(15):Restore colors:For f%=1 To 15:Read Col(f%):Next f%  
cls  
load bmp FN$  
x=0  
y=0  
For TNR=0 to num-1  
  
tn$="sprites3\SP3"+hex$(tnr,3)+".SPR"  
open tn$ for output as #1  
print #1,str$(W);",1,";STR$(H)  
for y1=y to y+H-1  
WT$=""
```

```

for x1=x to x+W-1
  C=Pixel(x1,y1):cl=0
  for n= 0 to 15:if C=col(n) then cl=n
Next

      wt$=wt$+hex$(cl,1)
Next
?#1, wt$
next
box x,y,w,h,,rgb (white)
close #1
inc y,h:if y>383 then y=0:inc x,w
next TNR

colors:
'--Цветовая схема в соответствии со Spritecolors
Data RGB(BLUE),RGB(GREEN),RGB(CYAN),RGB(RED)
Data RGB(MAGENTA),RGB(YELLOW),RGB(WHITE),RGB(MYRTLE)
Data RGB(COBALT) ,RGB(MIDGREEN),RGB(CERULEAN),RGB(RUST)
Data RGB(FUCHSIA),RGB(BROWN),RGB(LILAC)

```

Подобные программы занимают довольно много времени на каждый спрайт/тайл, поэтому их лучше всего запускать на MMB4W (ММБейсик для Windows

<https://www.geoffg.net/WindowsMMBasic.html#:~:text=MMBasic%20is%20a%20BASIC%20interpreter,quick%20scripting%20language%20for%20Windows> ).

Если вы хотите запустить его на PicoMite, измените «**load bmp FN\$**» на «**load image FN\$**».

### 3/ ПАМЯТЬ

MMB4W не поддерживает БИБЛИОТЕКУ, создание двоичного файла должно выполняться на самом PicoMite.

Чтобы загрузить файл **xxxx\_csub.bas** в память и сохранить его в библиотеке, вам потребуется программная память как для шестнадцатеричного текстового представления, так и для двоичного результата в ОЗУ. Поэтому вы можете обрабатывать только файлы **xxxx\_csub.bas** размером 60 КБ в программном пространстве размером 100 КБ. Именно поэтому тайлы и спрайты распределены по нескольким папкам, поэтому файлы **xxxx\_csub.bas** соответствуют этим требованиям.

## Заключение

Как видно из вышеизложенного, работы по подготовке довольно много, поэтому начинать предпочтительнее с четко определенного набора графических элементов. Чтобы сделать это для 400 элементов, потребуется не мало времени. Но наградой станет быстрая графическая система. В PETSCII графика занимает всего 30% игрового цикла.

Если вы новичок в этой деле, начните с графического набора из нескольких графических элементов, пока не освоите процесс.