КАК ЗА МЕСЯЦ ВЫУЧИТЬ ZX BASIC И НАПИСАТЬ ДЕМУ ИЛИ MAKING OF “WHITE STARS”

Если ты ничего не чувствуешь, значит ты умер.

Напомню, 7 сентября 2015 года, на hypr.ru было объявлено забавное и необычное демокомпо — BASE-X. Необычность его в том, что принимаемые на компо демо должны быть написаны исключительно на zx basic. Интерес к бейсику есть, о чём говорит не нулевое количество демо, написанных на бейсике и возникновение компо — это весьма закономерное событие.

Мне всегда был интересен бейсик с точки зрения простоты понимания его непрограммистом. А уж что может быть проще, чем реализация бейсика на zx spectrum? Фактически, не являясь программистом, взять и что-то написать на бейсике — довольно просто и я уже не раз предпринимал попытки (EVM - 3bm openair 2011, 3bm heartbeat - 3bm 2015) что-то сделать по-быстрому, используя обрывки воспоминаний из детства и первый попавшийся гаджет под рукой — Nintendo DS с эмулятором ZX Spectrum. Но ведь всегда хочется чего-то большего и цельного, а не просто попытка попищать бипером и показать что-то на экране. А после выхода таких замечательных продуктов, как Silabba, Back to basics и Silent attraction зудеть стало еще сильнее...

Пора браться за дело, подумал я, и открыл предложенный diver4d Basin — редактор-эмулятор, заточенный под бейсик. Это стало отправной точкой разработки и изучения. Скажу больше, Basin и его форк BasinC с незначительными изменениями, являются по сути IDE для разработки программ на zx basic. Используя только Basin можно и написать программу и выловить баги и нарисовать графику и выпустить цельный и законченный продукт.

Все мои знания бейсика ограничивались несколькими операторами: FOR NEXT PRINT PLOT DRAW BEEP POKE IF THEN GO TO GO SUB LET PAUSE и вроде бы всё... Ну что же, подумал я, и с этим можно начинать.

С чего начинать дему? Конечно же с приветов!

Первым делом была попытка написана часть “Rainbow hello”.
Изначально задумка была такой: выводим строку черным по черному, потом зажигаем атрибутами в несколько проходов познакоместно, чтобы проявление было постепенным, радужным.

Первично выводил строку PRINTом, потом в цикле прокручивал атрибуты в экранной области. Оказалось, что выводить строчки отдельными PRINT-ами довольно занудно и громоздко, а крутить атрибуты в цикле всё равно медленно. Вывод работает, цвета меняются, но нет изюминки и как-то обычно. Полез в справочник (а в Basin есть встроенный и очень неплохой HELP), глядеть как работают строковые переменные и оператор DATA.

Вот отрывок из варианта до сборки:

```
20 FOR a=1 TO 3
40 READ a$
50 LET l=LEN a$: LET x=INT ((32-l)/2)
60 FOR f=0 TO 8: PRINT BRIGHT 1; OVER 1; INK f; PAPER 0;AT 11,x;a$: NEXT f
100 DATA ''zzz'',''xxx'',''yyy''
```

Используя данные, а не целую строку в PRINT у меня возник вопрос, а как вывести надпись красиво?

Сначала я думал хранить координаты вывода вместе со строковыми данными или генерировать по ходу с помощью RND, от RND я сразу отказался т.к. во-первых это достаточно медленно, а во-вторых есть шанс холостых циклов по условию, если строка не поместится полностью до края экрана. Сделав вариант с хранением координат я понял, что добавление или удаление данных становится очень неприятным и рутинным, т.к. всегда нужно пересчитывать и прикидывать координаты для вывода, да и программа становится более громоздкой. Тогда было решено, что выводиться строка будет в середину экрана и вычисление координаты вывода сделать достаточно просто. Тут я снова полез в HELP искать пояснения про функцию LEN, которая вычисляет длину строковой переменной. Теперь, если мы из ширины экрана в знакоместах вычтем половину длины строки, то получим x-координату для вывода строки. Отлично! Вот так код становится дизайном, а не дизайн кодом, подумал я. И для бейсика это достаточно важная часть выбора т.к. мы ограничены в быстродействии и памяти. Т.е. важен компромисс между “БОЖЕНЬКИ, КАК КРАСИВО!” и “ОХТЫЖ КАК БЫСТРО!”.

Задача минимум была решена, строчки выводились исправно. Тут я подумал, что из этого можно сделать неплохой мессенджер, если смещать еще и y-координату вывода строчек, что и было сделано позже и оформлено в подпрограмму со своими входными данными, но для этого еще не раз пришлось заглянуть в шпаргалку по бейсиковским операторам.

Теперь возникла задача максимум, нужно как-то разнообразить наши приветы, иначе это не приветы, а просто мессенджер (хоть он еще был не реализован, но в голове он уже был отдельной подпрограммой). В банке знаний из визуальных альтернатив были лишь PLOT и DRAW и я начал экспериментировать.

Изначально, идея была в том, что снизу экрана из точки появляется квадратик, двигается к своему законному месту, разрастается к центру экрана в размер нашей строки, появляется, а за ним затухает хвост. На деле оказалось достаточно сложным для меня в реализации и очень медленным. Тогда я решил сделать обрамление сверху и снизу нашей надписи с зависимостью от длины выводимой строки и раскрасить в разные цвета. То что получилось меня вполне устроило и выглядело на удивление очень свежо для бейсика.

Единственный момент, который меня беспокоил — неравномерность по времени вывода линий в зависимости от длины, т.е. короткие сообщения и короткие линии выводились быстрее, а длинные медленнее, тогда я добавил паузу с зависимостью от длины выводимой строки:

```
146 PAUSE (50-(LEN a$))/3
```

Эффект был готов. На него по времени (вместе с освоением Basin и листанием HELPa) ушла наверное неделя =) А чистого времени, наверное часа 4 не меньше.

В принципе, подключив музыку на прерываниях уже можно было бы и успокоиться, но посмотрев на объем в 1к со свободными 39к — это было бы неразумно и глупо т.к. еще было достаточно времени, чтобы придумать и написать что-то еще, да и интерес разгорался всё больше и больше. Тем временем, смотря на свободное пространство в памяти, подумалось, что неплохо было бы и графику в дему добавить. Но в этот момент я даже не представлял, каким образом и как это сделать. Конечно можно было бы поглядеть на реализацию вывода графики в других демах и покопаться в исходном коде, но попытавшись залезть в исходники “b2b” и окончательно запутавшись в листинге я бросил это дело после 15 минут.

Потом пошли эксперименты с рисованием квадратиков по PLOT DRAW, вычисления SIN COS, попытка поиграться с полярными координатами:

```
5 LET x=1: LET dx=10+(2*(-10*RND))
10 PLOT 255/2,88
20 LET yx=15*(RND)
30 LET x=x+yx
40 DRAW -dx,yx
45 IF x<50 THEN GO TO 20
50 GO TO 510 LET r=10: LET s=.1
15 LET s=s+.1
20 FOR t=0 TO 2*PI STEP s
25 REM LET r=changes for radius
30 LET x=10+r*COS (t)
40 LET y=16+r*SIN (t)
50 PRINT AT 21,x;''********''
55 PRINT
60 NEXT t
70 GO TO 15

5 LET x=1: LET dx=10+(2*(-10*RND))
10 PLOT 255/2,88
20 LET yx=15*(RND)
30 LET x=x+yx
40 DRAW -dx,yx
45 IF x<50 THEN GO TO 20
50 GO TO 5
```

Ну, и так далее...

Из всех попыток сделать что-то стоящее я понял одно: нужно как-то выкручиваться из ситуации тормознутости бейсика. Либо прекалк, либо зрительный обман. Я прикинул, что самый быстрый вывод на экран — это PRINT символов в знакоместа, значит нужно воспользоваться этим максимально. Выводить весь экран из буфера нереально, уже очень медленно, значит нужно использовать последовательный, выборочный вывод и элемент случайности. Но, чтобы картинка не выглядело статикой нам нужно очень быстро скроллировать экран и у нас есть такая возможность в штатном бейсике, но мешает одна особенность. Когда места на экране не хватает, бейсик спрашивает пользователя нужно ли продолжать прокрутку экрана “Scroll?”. К счастью, у нас есть возможность продлить себе удовольствие и скроллировать за раз до 255 строк, а помогает нам в этом изменение системной переменной.

```
POKE 23692,255[SCR_CT 23692]

POKE 23692,255
```

Вот на этом трюке я и построил практически все свои эффекты. Признаю, что выглядит дема от этого очень одинаково, но на тот момент мне нужно было набрать необходимый набор эффектов, чтобы собрать хоть что-то более-менее смотрибельное на 2-3 минуты.

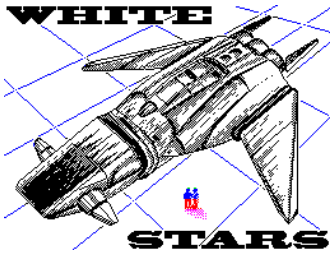
Далее, я стал потихоньку писать отдельные программы, чтобы был достаточный набор и не задумывался на том этапе о самом важном в деме — концепте =) Нет, конечно же у меня всегда были мысли о своей деме про космос, про роботов, про человечество, про сознание и осознание себя во вселенной, но я никогда не думал об этом в контексте бейсик-демы. Тем не менее, набрав пяток эффектов я решил, что пусть давняя мечта воплотится в этом продукте. Нечего протухать и умирать в забвении.

Так, неожиданно, появилось старое/новое название для демы “WHITE STARS”. Почему старое? Потому что я уже начинал рисовать когда-то графику, но большая часть была потеряна и проект остановился. Из сохранившегося осталась всего одна картинка, которая была выставлена на ZX AAA demo compo 2013:

И заставка, которая, вроде бы еще нигде не светилась, но уже не подходила по задумке:

Теперь нужно было рисовать новую заставку как минимум и дополнительную графику в дему как максимум. Если с заставкой было все более менее ясно — грузим в самом начале демы по LOAD”picture”SCREEN$, то с выводом графики я еще ничего не знал, но решил, что графика должна быть обязательно. Так, параллельно с написанием эффектов я начал рисовать заставку. И раз уж будем

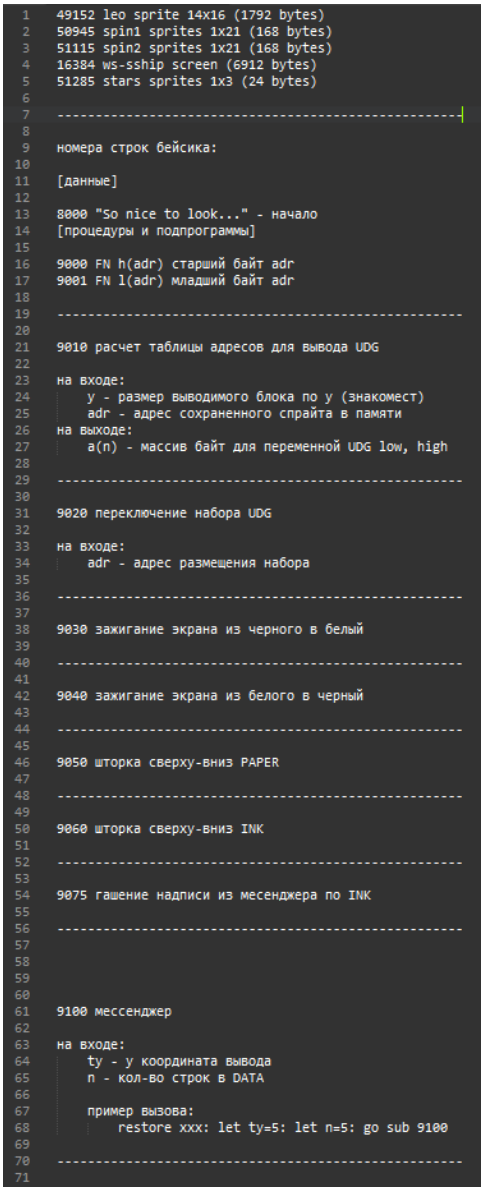
делать отсылки к космосу, то будем рисовать космический корабль. В конечном итоге я оставил рамку с погашенной яркостью для придания большего объема и глубины.



В конечном итоге у меня получилась примерно вот такая картина с набором простых бейсик программ:

calculate bytes	bas	274 байт
color lines	bas	2,3 Кб
color tunnel	bas	799 байт
color tunnel 2	bas	0,9 Кб
decrunch	bas	643 байт
dna draw	bas	502 байт
double height chars	bas	0,9 Кб
draw	bas	315 байт
draw 2	bas	459 байт
draw 3	bas	528 байт
font adr calculator	bas	296 байт
messenger	bas	708 байт
moving lines	bas	420 байт
new draw	bas	566 байт
polar	bas	404 байт
quadra	bas	2,1 Кб
quadra 2	bas	3,5 Кб
rainbow hello	bas	1,3 Кб
rainbow wave	bas	1,7 Кб
rainbow wave v2	bas	1,8 Кб
singularity	bas	2,5 Кб
sliceBAS		1,4 Кб
stars poetry	bas	1,2 Кб
stars poetry 2	bas	1,1 Кб
star vector 2	bas	1,8 Кб
star vectors	bas	1,1 Кб
text scroll	bas	393 байт
two byte adress calculator	bas	290 байт
wonder square	bas	1,7 Кб

Наверное, достаточно скучно будет разбирать далее всю дему по полочкам, как первый написанный эффект “Rainbow hello”, т.к. дальше уже появлялся опыт и ощущение чего-то непостижимого терялось со временем, но хотел бы пояснить некоторые самые интересные моменты для заинтересовавшихся.

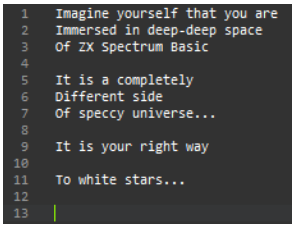


```
49152 leo sprite 14x16 (1792 bytes)
50945 spin1 sprites 1x21 (168 bytes)
51115 spin2 sprites 1x21 (168 bytes)
16384 ws-sship screen (6912 bytes)
51285 stars sprites 1x3 (24 bytes)

-----------------------------------------------------

номера строк бейсика:

[данные]

8000 "So nice to look..." - начало
[процедуры и подпрограммы]

9000 FN h(adr) старший байт adr
9001 FN l(adr) младший байт adr

-----------------------------------------------------

9010 расчет таблицы адресов для вывода UDG

на входе:
    y - размер выводимого блока по y (знакомест)
    adr - адрес сохраненного спрайта в памяти
на выходе:
    a(n) - массив байт для переменной UDG low, high

-----------------------------------------------------

9020 переключение набора UDG

на входе:
    adr - адрес размещения набора

-----------------------------------------------------

9030 зажигание экрана из черного в белый

-----------------------------------------------------

9040 зажигание экрана из белого в черный

-----------------------------------------------------

9050 шторка сверху-вниз PAPER

-----------------------------------------------------

9060 шторка сверху-вниз INK

-----------------------------------------------------

9075 гашение надписи из месенджера по INK

-----------------------------------------------------




9100 мессенджер

на входе:
    ty - y координата вывода
    n - кол-во строк в DATA

    пример вызова:
        restore xxx: let ty=5: let n=5: go sub 9100

-----------------------------------------------------

```

Самый большой мой промах был в том, что я изначально не стал вести записи того, какие участки программ являются подпрограммами или некими функциями, я просто копировал нужные куски в новую программу и даже не задумывался о процессе сборки в будущем. Но! Хорошо, что я занялся этим перед окончательной сборкой и когда еще не все эффекты были окончательно дописаны.

Вот примерно так выглядел мой док в процессе разработки.

Карта адресов где лежит графика, описание функций и подпрограмм с кратким описанием работы. Для подпрограмм — данные на входе и на выходе (если есть). Также, в отдельных текстовых файлах были записи по текстам для вывода в деме. Но тексты менялись по ходу написания в зависимости от количества необходимых строк для вывода или от обстоятельств, касающихся концепта демы.



```
Imagine yourself that you are
Immersed in deep-deep space
Of ZX Spectrum Basic

It is a completely
Different side
Of speccy universe...

It is your right way

To white stars...

```

Вот, кстати о концепте... По ходу написания как таковой концепт возникал, потом перерождался во что-то иное, потом возникал новый от того, что появлялись новые возможности и новые знания. Например, с того момента, как я вывел на экран свой первый спрайт, в голове возник очередной концепт =)

Но мне не хотелось отказываться от названия демы - “WHITE STARS”, она как заноза, мешала мне много лет, как незавершенный гештальт, не позволяла куда-то двигаться и что-то делать. Будь что будет, подумал я тогда. Будет каша и солянка, значит пусть будет так. Будет моя первая большая бейсик-каша.

Первым шагом, для сближения с графикой, естественно было знакомство с UDG. Сначала рисование во встроенном в Basin редакторе UDG с последующим экспортом прямо в тело программы строк DATA с данными символов. (В Basin это сделано очень удобно). И это мне очень помогло на первом этапе.

В дальнейшем оказалось, что переключать наборы графики можно всего лишь меняя системную переменную.

```
POKE 23675,low: POKE 23675,high[UDG 23675] (2)

POKE 23675,low: POKE 23675,high
```

Собственно, для расчета этих двух байт я быстренько написал подпрограмму:

```
10 INPUT ''adr='';adr
30 LET h=INT (adr/256)
40 LET l=adr-(INT (adr/256)*256)
50 PRINT ''Font adr='';adr: PRINT ''low byte='';l;'' high byte='';h
```

С её помощью я рассчитывал адреса и в своей основной программе переключал начало набора парой POKE.

Может возникнуть вопрос, а зачем же переключать не наборы, а смещать начало набора? Очень просто, таким образом, мы впечатывая в цикле один и тот же символ UDG можем менять отображаемую этим символом графику т.к. начало набора стоит уже с другого адреса.

Поясню:

PRINT AT x,y;“/a”
POKE
PRINT AT x,y;”/a”

В данном примере оператор PRINT будет выводит разную графику по координатам x,y.

Такое положение вещей мне очень понравилось и я начал думать над тем, как быстро вывести большой блок графики на экран. Всего в UDG нам доступен 21 символ в наборе т.е. мы можем вывести без особых заморочек блок графики 168x8 пикселов всего лишь одним оператором PRINT. Переключая наборы, мы можем в цикле вывести любой спрайт с размерами 21x21 знакоместо или 168x168 пикселов. Именно с задачи вывода максимального блока графики я и начал свои исследования.

Первой мыслью было: рассчитать табличку адресов, положить в память, процедурой вывода доставать из памяти адреса и переписывать значения в системной переменной в цикле, выводя при этом строку UDG.

Мысль была хорошая и чёрт меня дёрнул, написать расчёт таблички на ассемблере, уж если погружаться с головой в программирование и кодинг, то с головой =) Со второго раза у меня даже получилось написать вот это:

```
ld de,49152+6144+1
ld hl,49152
ld bc,168

DUP 21
ld a,l
ld (de),a
inc de
ld a,h
ld (de),a
inc de
add hl,bc
EDUP

retorg 30000

ld de,49152+6144+1
ld hl,49152
ld bc,168

DUP 21
ld a,l
ld (de),a
inc de
```

```
ld a,h
ld (de),a
inc de
add hl,bc
EDUP

ret
```

И оно даже правильно построило мне нужную таблицу и сразу же заработал вывод графики из бейсика.

Единственная проблема, во встроенном в Basin ассемблере нет DUP-EDUP и пришлось раскрыть большую простыню, чтобы не использовать сторонние ассемблеры.

Изначально я планировал хранить графику и таблички вывода в памяти вместе, т.е. [GFX DATA][TABLE DATA],
памяти на это расходуется не много: 21*2=42 байта для вывода максимального блока графики. Но, это очень неудобно, когда графика меняется, когда в любой момент потребуется перенести графику в другое место. В моём случае потребуется постоянно пересчитывать таблички. Если бы у меня изначально был чёткий план по деме, то вполне возможно, я бы и оставил всё в памяти и сильно ускорил вывод больших блоков графики т.к. не пришлось бы постоянно рассчитывать адреса из бейсика. А мы простых путей не ищем =) В дальнейшем я отказался от сохранения табличек, в обмен на возможность быстрого перемещения данных по памяти. Почему именно так? Ответ простой, когда писалась дема, еще не было готового решения с AY-плеером, не было музыки и не было четкого представления сколько займёт дема после сборки и сколько еще графики будет в деме... (даже когда пишутся эти строки — 3 ноября 2015, нет еще готового плеера и дема собрана только наполовину)

Как можно заметить, расчёт младшего и старшего байта адреса я оформил в функции бейсика и сделал процедуру расчёта таблички — строка 9010, и переключения набора символов — строка 9020.
На входе в подпрограмму рсчёта: adr - адрес графики в памяти, y - размер выводимого блока по вертикали. Табличка сохраняется в массив a(n).

```
9000 DEF FN h(n)=INT (n/256)
9001 DEF FN l(n)=n-(FN h(n)*256)

9010 FOR f=1 TO y*2 STEP 2:
   LET a(f)=FN l(adr):
   LET a(f+1)=FN h(adr):
   LET adr=adr+(x*8):
   NEXT f:
   RETURN
9020 POKE 23675,FN l(adr):
   POKE 23676,FN h(adr):
   RETURN      DIM a(42)
```

```
9000 DEF FN h(n)=INT (n/256)
9001 DEF FN l(n)=n-(FN h(n)*256)

9010 FOR f=1 TO y*2 STEP 2:
     LET a(f)=FN l(adr):
     LET a(f+1)=FN h(adr):
     LET adr=adr+(x*8):
     NEXT f:
     RETURN
9020 POKE 23675,FN l(adr):
     POKE 23676,FN h(adr):
     RETURN
```

Достаточно топорное и тормозное решение, но оно продиктовано сложившейся ситуацией.

Ну, а сам вывод спрайта выходит довольно простым:

```
1005 LET y=16: LET adr=49152
1010 GO SUB 9010
1015 FOR f=1 TO y*2 STEP 2
1020 POKE 23675,a(f): POKE 23676,a(f+1)
1025 PRINT ''                  \a\b\c\d\e\f\g\h\i\j\k\l\m\n''
1030 NEXT f
```

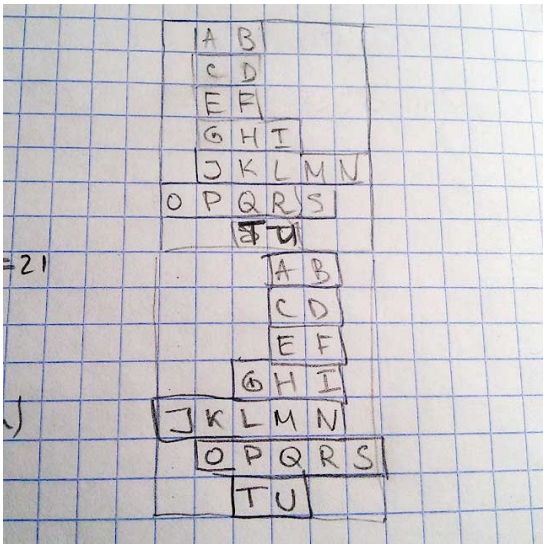
В переменную Y - размер блока графики по вертикали, в переменную ADR - расположение графики в памяти. Для простоты и быстроты координату X регулируем пробелами в операторе PRINT.

На этом для себя тему с выводом графики я решил закрыть. Есть еще вариант с печатью из простого набора символов с тем же самым переключением 768-байтных банков графики для шрифта. Но я не планировал вывод полноэкранной графики и к тому же будет биться набор символов, что усложнит редактуру исходного кода, а для меня это критично, я только учусь!

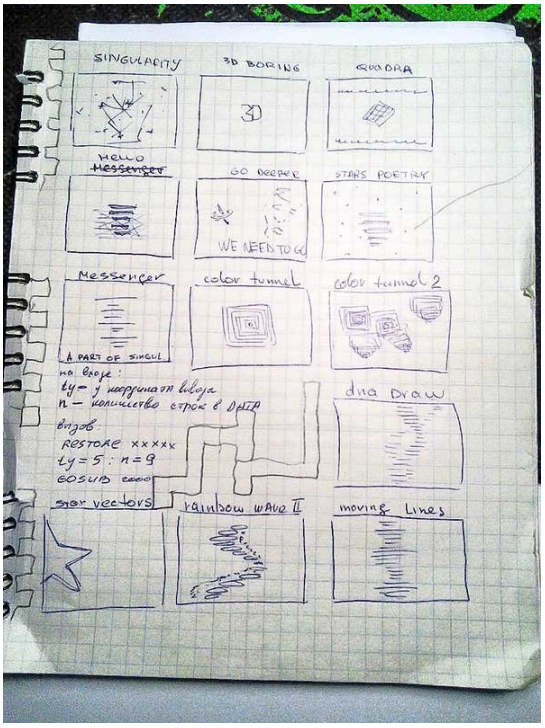
Спрайт андроида - это не вошедшая в zx spectrum часть картинка демы Synchronization. Планировалась еще и анимация, но я понял, что с анимацией-то вообще ничего не успею.

Спрайт “Лео” и парочка волчков. Снизу — один уже разложен по блокам 8x8 правый.

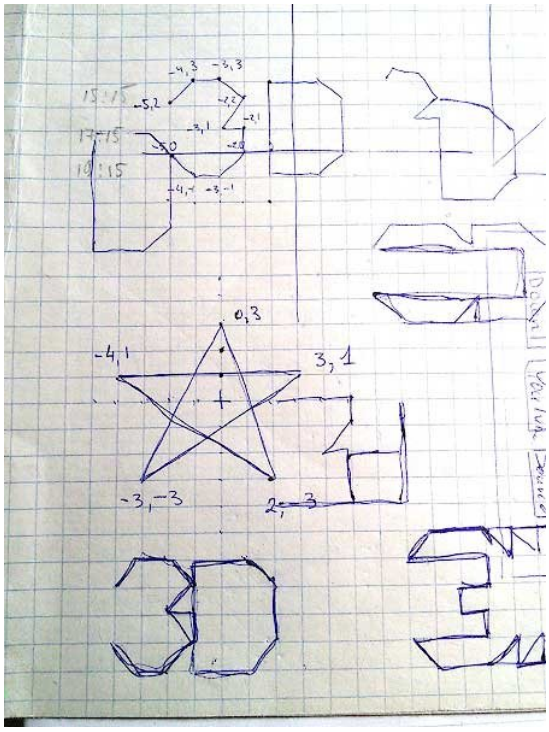
А вот так выглядит раскладка спрайтов левого и правого волчка. Используется ровно по одному набору UDG.



Еще немного ламповости добавлю вот этими фотографиями. Иногда на бумаге думается и размышляется гораздо продуктивнее и вот эта ностальгия по детству и исписаным тетрадкам.

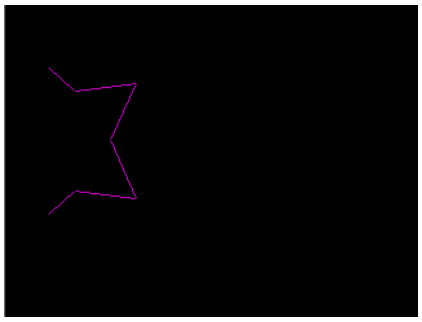


Вот, на этой фотографии видно как я выбирал объект для 3d части в деме.



Почему 3d и зачем в бейсик деме?

Потому что было интересно, насколько это возможно и насколько это медленно. Опять же выбор у нас небольшой — это либо прекалк, либо честное или не очень честное 3d. Я начал с прекалка. Всё просто, закидываем координаты в массив или в DATA или кидаем в память, читаем данные, выводим. Чем больше вершин обрабатываем, тем больше тормоза. Естественно, посчитать нужно так, как это будет выводится на экран и управлять выводом мы потом уже не сможем.

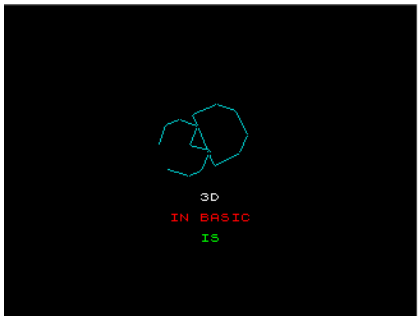


Я взял половину звезды и планировал, что она будет крутиться, наполовину выходя за экран. И когда я уже рассчитал все координаты, понял, что количество вершин у меня меняется. Это немного усложнило вывод, но всё получилось.

```
5 DIM z(64)
6 LET r=100
10 PAPER 0: INK 7: BORDER 0: CLS
20 FOR n=1 TO 64
```

```
30 READ z(n)
40 NEXT n
100 LET n=1: LET m=14: GO SUB 2000: GO SUB 2000
110 LET n=17: LET m=30: GO SUB 2000: GO SUB 2000
120 LET n=33: LET m=46: GO SUB 2000: GO SUB 2000
130 LET n=49: LET m=62: GO SUB 2000: GO SUB 2000
999 GO TO 100
1000 DATA 0,48,44,27,38,76,72,112,24,120,0,164,0,164,0,164
1010 DATA 0,38,20,56,68,50,48,96,68,140,20,134,0,152,0,152
1020 DATA 0,26,24,70,72,80,38,115,45,164,0,144,0,144,0,144
1030 DATA 0,38,16,23,25,72,68,96,25,120,16,168,0,152,0,152
2000 INK 2+INT (0.02*PEEK r)
2003 FOR f=n TO m STEP 2
2010 PLOT OVER 1;z(f),z(f+1)
2020 DRAW OVER 1;z(f+2)-z(f),z(f+3)-z(f+1)
2025 LET r=r+1
2030 NEXT f
2040 RETURN
```

Теперь немного о “настоящем” 3d. “Настоящее” я взял в кавычки потому, что вместо SIN-COS при преобразовании поворота я использовал вычисленные значения.

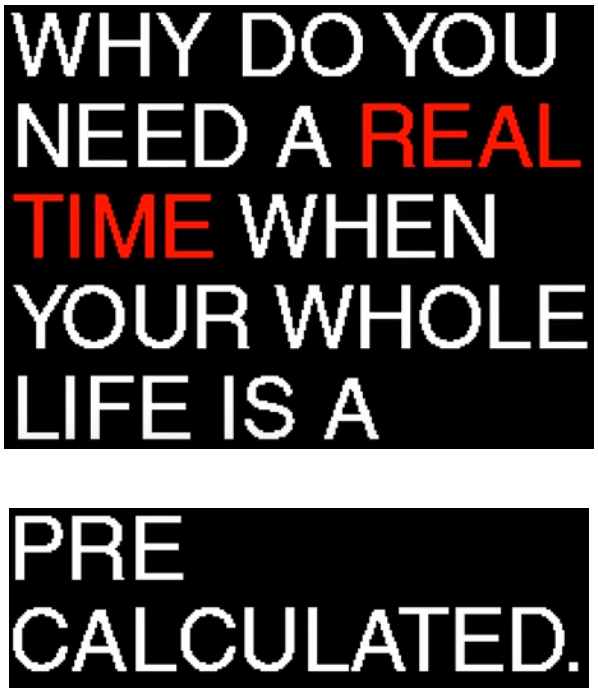



```
10 DIM z(32): DIM c(32)
20 LET r=1
30 PAPER 0: INK 7: BORDER 0: CLS
40 FOR n=1 TO 32
50 READ z(n): LET z(n)=z(n)*10: LET c(n)=z(n)
60 NEXT n
64 RESTORE 115
65 FOR l=1 TO 24
```

```
66 POKE 23692,255: READ l$: PRINT BRIGHT 1;AT 21,16-(LEN l$)/2;l$: PRINT :
PRINT
70 GO SUB 120
80 GO SUB 250
90 GO SUB 190: GO SUB 290
100 NEXT l
105 STOP
110 DATA -3,2,-2,3,-1,3,-0,2,0,3,2,3,3,2,3,0,2,-1,0,-1,0,2,-1,1,0,0,-1,-1,-2,-1,-3,0
115 DATA ''3D'',''IN BASIC'',''IS'',''REAL THING'',''BUT'',''SO
BORING'',''LET'S'',''TRY'',''SOMETHING'',''FASTER'',''...'','''',''''',''''',''''',''''',''''
119 REM risuem
120 INK r
130 FOR f=1 TO 30 STEP 2
140 PLOT OVER 1;128+INT (z(f)),88+INT ((z(f+1)))
150 DRAW OVER 1;z(f+2)-z(f),z(f+3)-z(f+1)
160 LET r=r+1: IF r>7 THEN LET r=1
170 NEXT f
180 RETURN
189 REM poworot
190 FOR f=1 TO 32 STEP 2
200 LET rx=INT (z(f)*0.97)-(z(f+1)*0.20)
210 LET ry=INT (z(f)*0.20)+(z(f+1)*0.97)
220 LET z(f)=INT (rx*1.05): LET z(f+1)=INT (ry*1.05)
230 NEXT f
240 RETURN
249 REM perebros massiva
250 FOR f=1 TO 32
260 REM READ z(f)
270 LET c(f)=z(f)
280 NEXT f: RETURN
289 REM stiraem staroe
290 INK r: FOR f=1 TO 30 STEP 2
300 PLOT OVER 1;128+INT (c(f)),88+INT ((c(f+1)))
310 DRAW OVER 1;c(f+2)-c(f),c(f+3)-c(f+1)
320 LET r=r+1: IF r>7 THEN LET r=1
330 NEXT f
```

```
340 RETURN
```

Ужасно медленно. Больше не делайте как я, не надо честного или “честного” 3d в бейсике. Прекалькулируй это! Ну, вы помните да?



Вот это пожалуй и все самые интересные моменты написания демы (хотя, как знать, дема-то еще не написана, когда пишутся эти строки).

Из всего вышесказанного я прямо сейчас могу сделать несколько выводов и дать пару советов:

1. В самом начале взять лист бумаги и написать один абзац текста с подзаголовком: “О чём эта дема:”
2. С самой первой секунды работы над демой (не имеет значения работаете вы в одного или в команде) ведите диз. доки, тех. доки, ТЗ, любые описания работы и аккуратно складывайте в папочку “СУПЕРДЕМА”. Ваша голова и головы всех, кто в проекте не удержат весь поток информации, который будет генерироваться или всплывать по ходу. Если вы в команде, выберите главного, который будет принимать окончательное решение по всем вопросам. Нет главного — нет демы. В случае неудачи пусть все шишки достанутся ему ;)
3. Не переоценивайте свои возможности. Времени всегда не хватает.
4. Советуйтесь!
5. Начинайте с простого. Усложняйте постепенно. И будьте готовы переписать всё заново, 2 раза.
6. Если можно сделать просто, делайте просто. Если можно обмануть, обманывайте.
7. Получайте удовольствие от процесса. Это самый главный пункт. Если при написании вы получаете положительные эмоции, поверьте, кто-то потом порадуется точно также.

Буду рад ответить на возникшие вопросы в комментариях, а пока, пойду дособирать эту дему!

P.S. сразу предсказывая вопрос про “НЕ ПРОГРАММИСТ НИ РАЗУ”, хотя упомянул про тетрадки исписанные в детстве... так вот, то было в детстве и давно забыто. Было немного бейсика и совсем чуть-чуть ассемблера — это правда.