"Kylie Minogue"

Vice snapshot with CCS64 palette

Made with the GIMP from a KM photo
and converted to C64 160x200
Multicolor Mode Bitmap
by Stefano Tognon
in 2002

"Are you ready for this new magazine?"
...

Team Ice Free Software Group

SATAin 1
version 1.00
21 September 2002

General Index

Editorials

Stefano Tognon <ice00@libero.it>

Welcome, SID fans.

This is the first issue of a Sid related magazine.

The idea of this magazine born with the porpoise to give some feedback about sid activity to other sid people. I hope that other people may contributed to the magazine.

The magazine will have a *news* section that reports what happens from an issue to another in the sid activity: tools, programs, events, music, discussion... and all that can be considerate interesting to know. The inserted news are to be considerate evergreen, so they can be always useful.

The rest of the magazine will have various articles that speak about sid programming technique, players, emulation, hacking, practical experience... and all that are around the sid.

I hope that you want to contributed by give your articles or opinions. If you are/were a composer, a programmer like me, a sid listener, than let your experience jumps out of you and go to be a part of us!

This magazine is write with OpenOffice for bigs reasons:

- I want an open archive format that are to be available on most of the platforms.
- I want a not text based format (so images can be added).

Even if I can choose other tools, the less free time has choose for me!

However, the magazine will be available in OpenOffice sxw file and in pdf printable version.

Well, how about the next issue?

As I dedicate most of the free time to the Sid, I'm writing other articles right now, but there are so many C64 activity that I'm in that I cannot say when the next issue will be out. You have just to be around when this happen :)

News

Well, as this is the first issue there are about 20 years of sid news to add!!! So I insert some various stuff:

1. HVSC version 5.0
2. CGSC version 1.9
3. Sidplay2/w
4. Goattracker 1.33
5. PSID64 0.4
6. Xsisplay 1.6.5pre15

HVSC version 5.0

Released on August 17, 2002 this is an epochal version: it uses the new sid file format PSIDv2NG!

Now there are about 18,616 SID file into the collection, and in this update 32 there are:

0 new SIDs	:((
346 fixed/better rips	:))
2 repeats/bad rips eliminated	
122 SID credit fixes	:))

In the next update (33, schedule for end October) there will be some real C64 tunes!

CGSC version 1.9

The Compute's Gazette Sid Collection (http://www.c64music.co.uk/) is at version 1.9 and was released in December 2001.

In the collection there are:

- 5773 MUS
- 1233 STR
- 1523 WDS (including 73 with Extended Words)

You may use sidplay2/w, sidplay2 or xsidplay (compiled with libsidplay2) to listen to the stereo STR sid files. In the site you can also see how to made the VICE dual sid support to play stereo sid music.

Sidplay2/w

Released at September 15 2002, this is a must for a windows user. You can download this player from http://www.student.nada.kth.se/~d93-alo/c64/spw/

It is based onto the last libsidplay2 library (http://sidplay2.sourceforge.net/), so you can hear the best emulated sid sound ever, and is aligned with the new extension of sid file..

It runs even in Linux throw Wine!

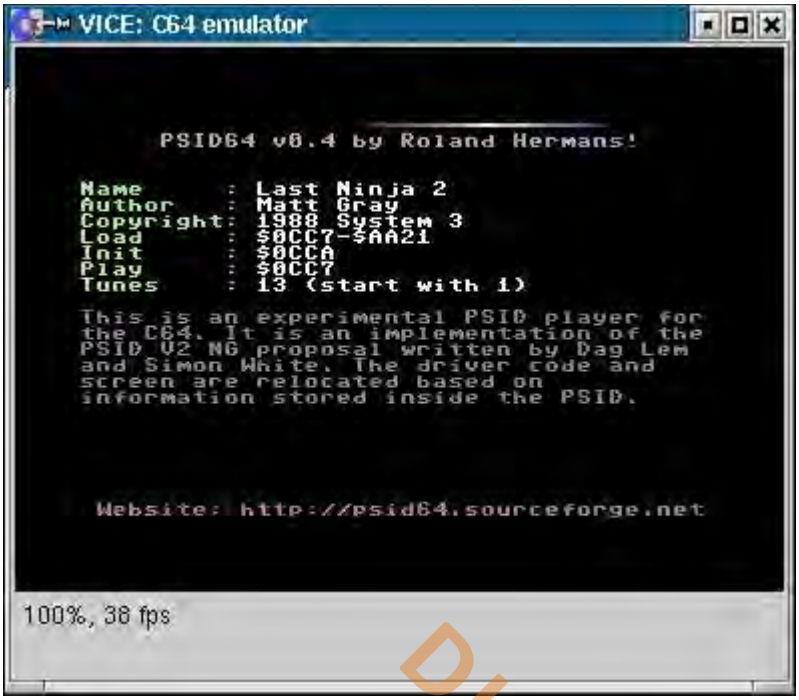
Goattracker 1.33

The music editor by Lasse Öörni for cross develop is now at version 1.33. From version 1.3 the 12 raster time lines is not be granted anymore, but the player is becoming more interesting and from version 1.24 it supports even multispeed tunes. Now, using SDL it run even in Linux.

Download GoadTracker from: http://covertbitops.cjb.net/tools/goattrk.zip

PSID64 0.4

PSID64 (http://psid64.sourceforge.net) is at version 0.4.



What is PSID64? Here the itself description:

PSID64 is a program that makes it possible to listen to your favorite SID music on a real Commodore 64 computer. It automatically generates a C64 self extracting executable from a PSID file. The executable contains the PSID data, a pre-relocated player and may also - if there is enough space available in the C64 environment - contain a demonstration program with information about the PSID file.

Even if here I put a VICE snapshot, I can grant that it runs perfectly in my C64s.

XSIDPLAY 1.6.5pre15

XSidplay (http://www.geocities.com/SiliconValley/Lakes/5147/sidplay/) the Unix player is now at version 1.6.5pre15.

It runs with libsidplay-1.36.57 or libsidplay2 CVS.

The player supports the new PSID2NG, but libsidplay1 may crush when listening to C64 specific true RIP as will come in the future HVSC upgrade.

Some discussion is being done for this, so probably a solution will be found until the next HVSC upgrade, otherwise listen to old PSID rips or use the libsidplay2 engine.

Luca Carrafiello Interview!

by Stefano Tognon

I'm very happy to start this first issue of SIDin with this interview to Luca/FIRE. First of all, because he is a Italian people like me, secondary because I like his music (Soulfixer is stellar - I buy a C64C recently only for listen this tune in the real 8580 chip!, and Central_Park_Loader_Y2K is one of the Matt Gray LN2 cover very innovative).

Now, here the interview that occurs with some emails in September of this year.

Hello, Luca,
Take some words to describe us a little about you: where do you live, what do you do, ...: just a presentation about your real life.

I'm Luca Carrafiello, aka Luca of Fantastic Italian Research Enterprise (FIRE) crew, and I'm 29. I'm graduated in Physics Chemistry and currently I'd started my Ph.D. here at the Florence's University. Though nowadays I'm living beside Florence, Italy, I came from Salerno, few kilometers south from Naples: Florence had begun my city only because I started studying here...
My life runs like a classic "single"'s way of life, with too much freedom and sudden amasses of spare time after a period of hard work, generally used for sustain my hobbies: Commodore, play Gameboy Advance's games (at the moment I'm playing "Super Robot Taisen R", and improving the linked GreatMazinger-MazingerZ weapons!), read lots of books (actually Johnathan Coe's), listening to relaxing music (Mike Oldfield, ISAN, Franco Battiato, Einsturzende Neubauten, Enya, Deep Forest, Michael Nyman, Wim Mertens...); and I like very much run in circles around Florence with my bike, pausing at every vintner in order to drink a glass of Chianti.

Tell us how is initiated your passion for the Commodore and the steps that you have accomplished during the years for reaching yours actual results (IMHO fantastic sid music)

Stefano, d'ya wonder to kill me with a blade forged with reminders? :D
Well, when I was 14 my parents got my request of buying an home computer, and they were so much surprised that they only spitted out money: I never asked for a toy or anything else before. I bought a Commodore Plus/4, 'coz I loved its design and its Basic seemed very user-friendly... But after few weeks frustration started: no hardware sprites, and, most of all, no SID!
Agh!
In those days, the only programs sources for C16 in Italy (and for it only, not Plus/4!) are the original games and the hacked games, sold in stands as tapes containing 6 of them; these hacked packs were generally released with a little newspaper. Once a beautiful day, in an issue of these papers I found an advertisement about Plus/4 only programs: gettin'contact with that man had taken me in the real Plus/4 scene, and, undirectly, in the C64 scene too.
I reached the Plus/4 scene in its best moment, at the beginning of 90s:
lotta crews, lotta stuff, lotta parties...and I got the desire of take part of. Thus, I started to draw logos, pics, chars...but I dreamt about composing SID tunes!
The Plus/4 sound's calvary starts with its poor 2-voiced squarewave/noise sound, but since the end of 80s, people had learnt how to convert SID data and play them on Plus/4, while a channel alternates 2 SID voices. 2 years after, the frq converter was supplanted, when possible, by the wave converter, which plays SID files with correct waveform, volume and SR.

Obviously, these emulations were far away from perfection, but I started to compose SIDs on a converted version of the Future Composer V2.1. Nowadays those tunes are included in the High Voltage SID Collection, but many of them sound really bad, 'coz on the Plus/4 I wasn't able to emulate filters, correct AD, sync, ring...
At a certain moment, many users had begun to buy a SIDcard, a sort of plug-and-play SID chip that allows the Plus/4 to run SID sounds. I bought it many years later from the original producer, in the 1999, and started composing on an improved FC's SIDversion. Finally, a cool Plus/4 user (Levente Harsfalvi, aka TLC/Coroners) converted on Plus/4 the Taki's SIDwinder V01.23 for the SIDcard; probably, only in that moment I'd really understood the real SID operation way, and so I had to learn again how to compose SIDs. With decent results, I hope.

Although I'm not a coder, and that's the real pity for me, I put some LDA, STA and PLP somewhere and my Plus/4 diskmagazine popped up. And I continue to release it. That's the reason because my musics are never multispeed ones: I have to use them abitually for the diskmag and can't manage a multispeed tune. And sometimes, I proclame some compos: the Plus/4 LogoCompo reached the 3rd edition.

Nowadays, I see that all the 8bit scenes are united under the same great family, and in many cases we can mind c64 sceners coding gameboy's demos, or ZX games... I only hope that trend will continue and grow. My eternal frief continues to be: if only I learnt assembly in past, now I was sustaining the scene by himself...bah, sh*t...

Now some quick final questions:

Real machine vs emulator: what do you think of?

The real machine remains the real machine: the best emulator ever should emulates the real one at 0.999%, but never at 100%.
Once said it, I see much Jules Verne's syndrome: early emulators were really far away from the real machine, and many users screamed about "shitty pc clones", but they're becoming something really closer to a C64 or ZX or whatever. E.g.: on the Plus/4 side, just during these days the digi emulation has become reality, and I jumped on my chair when I heard working waveconverter music from my PC running Minus4 V2.5 or YAPE V0.41.
Emulators are a therapy for the scene, but not the panacea.

What is the worst sid that you compose and the better one?

In order to answer this question, I must exclude all the musics composed directly on Plus/4 without SID support, although "Tubular Bells II Shake" tooks one month to be completed, and it's 13:30 ca. long. If you listen to it on a SID, real or emulated, it plays very bad, 'cause I badly tried to use the three default filtereffects offered by the old Future Composer; but surely it was a really hard work!
Thus, considering only music from 1999, I experimented very much the SID functions, just like a SID-rookie, but in the same time I vent all the ideas I wasn't able to apply without SID chip. Probably, the very first music that made me proud is the last I've composed, "Soulfixer", that made 2nd in the SidWine online compo... Unfortunately, I had to finish it with a certain hurry, so many ideas were truncated. The worst? All the others, 'coz I'm learning yet alot!

Who are your best sid authors?

There are too many hypercomposers, and a selection is very difficult. But I have a measure's unit: a musical composition can be enjoyable regardless to the instruments'quality, the use of filters and other crap. And a bunch of composers has caught very original and innovative ideas to put into a SID, some many more than others maybe...
Yes, surely, Dave Whittaker, Jeroen Tel, Rob Hubbard, Martin Galway, Drax, Matt Gray...but I like surprising melodies like "Zynaps prerelease" by Nigel Grieve, "Equinox" by Nick Jones, or "Death or Glory" and "Counterforce" by Jay Derrett: the cute music you don't expect!

What are the best sids ever in your opinion?

First of all, let be many people angry: I like multispeed tunes, but I consider them a sort of "subcategory", like digis, and I seriously think it would be good to create a multispeed compo instead of mixing them with "normal" singlespeed tunes (how racist!). E.g., "Strangers in the Flight" by Drax, "Gopho" by Goto80 and "GRG in Cyberspace" by Glenn Gallefoss are stunning! But my heart was taken out of this world by other musics: "Wizball" by Galway as first, then many others, like Matt Gray's "Last Ninja 2", Moppe's "Shadow of the Beast", Geroen Tel's "Cybernoid", MozIcArt's "Luminous", Zardax's "Eldorado Part1"...oohh, too many, too many, sorry! :D

Finally, many thanks for the time you give for this interview, but now there is a post scriptum question (yes, you are allowed to not answer :)):
Sid and girls: what to think?

Aha! I've got lotta experience in this combo! Girls always will appreciate your works on the SID chip: dunno why, but they're always so sensible about what they're listening, and girls instantly understand your effort to take out cool melodies from something that seems so toy-ish...
It's tested: you and the girl will move from the C64 desk directly on the bed, try it to believe it! ;)

Webography [from Luca]:

LoneNews archive:
http://plus4.emucamp.com/lone.htm

Plus/4 shmup project:
http://www.xeo3.com/

Plus/4 archive:
ftp://c64.rulez.org/pub/plus4

HVSC's hp:
http://home.freeuk.net/wazzaw/HVSC/indexhi.html

Iseq music composition program

Stefano Tognon <ice00@libero.it>

Steve Judd had make a new production: Iseq a new kind of music player for the C64. The reason for speaking here for this program is simple: it is very different respect all the other players around. It is very interesting to see how the community have saw this program by looking at the *comp.sys.cbm* threads about Iseq. After that we go into some technical stuff, by looking at the Steve Judd's Commando remix using Iseq.

Comp.sys.cbm Iseq threads

From:Robert Bernardo (rbernardo@value.net)
Subject:Iseq music composition program released
Newsgroups:comp.sys.cbm, alt.c64
Date:2002-06-09 23:28:35 PST

Steve Judd has released Iseq, a new music composition program for the C64 (see Steve's description below). Program, examples, docs, and screenshots are at www.ffd2.com/fridge/iseq

Truly,
Robert Bernardo
Fresno Commodore User Group
http://videocam.net.au/fcug

Iseq is, of course, a C64 music composition program, written totally from scratch (i.e. it isn't just a Tunesmith update). It has a lot of features but is fairly easy to use, with online documentation (as well as text stuff below). Feedback would be most welcome!

5/31/02 Page officialy goes online!

Main files:

* iseq10.zip Program, docs, examples.
* ISEQ1.D64.gz A .d64 of the binaries.
* iseq.docs Main documentation.
* iseq.tutorial Tutorial on getting started.

Tunes: (you can listen from BASIC using SYS 4102)

* snood.tune A tune written for the game "Blockhead's Revenge" (included in .zip above)
* examples.tune Some example instruments (note: some adapted from instruments written by daBlondie in Tunesmith, lo these years ago).

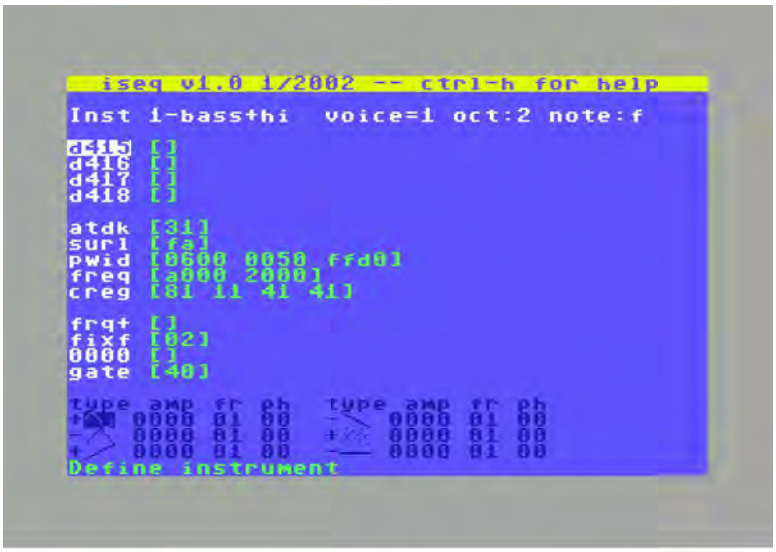
Source code:

The source is pretty big, so it might take a while to get up here. But if you email me I'd be happy to send it to you (does anyone even look at this stuff?!?).

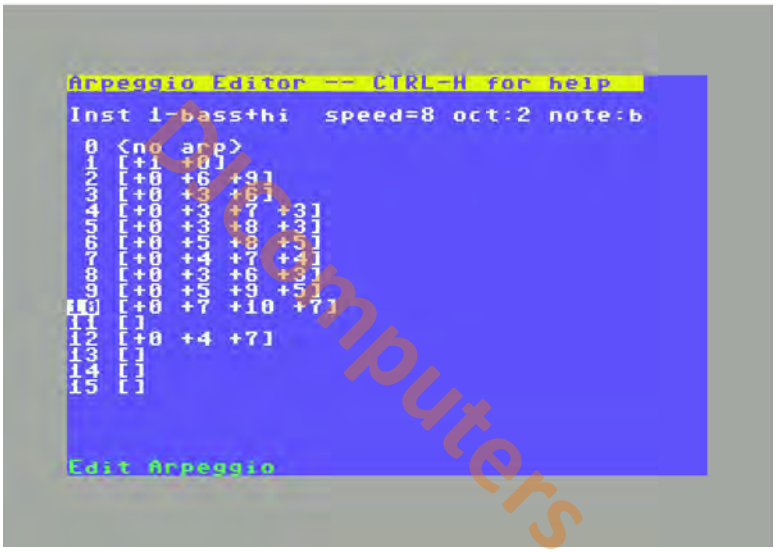
Some pictures:

The main editor. **Iseq allows multiple tunes, among other things.**

Online help is available for all screens.



The instrument editor. Instruments consist of sequence values or modulation schemes, and are compiled into the player.



Arpeggios can be of any speed or length, with both positive and negative values, etc.

Stephen L. Judd
sjudd(at)ffd2.com

From:Blackspawn (noone@home.net)
Subject:Re: Iseq music composition program released
Newsgroups:comp.sys.cbm, alt.c64
Date:2002-06-10 02:13:02 PST

Robert Bernardo schrieb:

> Steve Judd has released Iseq, a new music composition program for
> the C64 (see Steve's description below). Program, examples,
> docs, and screenshots are at www.ffd2.com/fridge/iseq

Hmm, I've only looked at the screenshots yet but I somehow doubt that anyone would really use that program. It looks much too complicated and spartan. IMO there are already much better tools for making music for the C64, like GoatTracker or OdinTracker for example...

/Blackspawn\

From:Agemixer (agespam@NOSPAM.japo.fi)
Subject:Re: Iseq music composition program released
Newsgroups:comp.sys.cbm, alt.c64
Date:2002-06-10 03:51:52 PST

On Mon, 10 Jun 2002, Blackspawn wrote:

> Robert Bernardo schrieb:
>
> > Steve Judd has released Iseq, a new music composition program for
> > the C64 (see Steve's description below). Program, examples,
> > docs, and screenshots are at www.ffd2.com/fridge/iseq
>
> Hmm, I've only looked at the screenshots yet but I somehow doubt that
> anyone would really use that program. It looks much too complicated and
> spartan. IMO there are already much better tools for making music for the
> C64, like GoatTracker or OdinTracker for example...
>
> /Blackspawn\

I checked it out and i must agree. I noticed it is too far from the common 'standards' of music player/editor. The user interface and keyboard logic is far from the common usage of most 'professional' kind of music editors around.

With first tests it hungs when adjusting the speed and the loader doesn't seem to work with Action Replay cartridge. The tempo change to CIA timers sounds funny, but it makes a lot of restrictions in demo and game needs. Thought, it is said in the docs it can be played with VIC's rasterrupt too.

I wouldn't recommend Iseq for any serious music making.

--
Agemixer/SCL

From:Steve Judd (sjudd@swcp.com)
Subject:Re: Iseq music composition program released
Newsgroups:comp.sys.cbm, alt.c64
Date:2002-06-10 17:41:22 PST

Hola,

In article <Pine.LNX.4.33.0206101321350.16410-100000@aapo.japo.fi>,
Agemixer <agespam@NOSPAM.japo.fi> wrote:
>

>On Mon, 10 Jun 2002, Blackspawn wrote:
>
>> Robert Bernardo schrieb:
>>
>> > Steve Judd has released Iseq, a new music composition program for
>> > the C64 (see Steve's description below). Program, examples,
>> > docs, and screenshots are at www.ffd2.com/fridge/iseq
>>
>> Hmm, I've only looked at the screenshots yet but I somehow doubt that
>> anyone would really use that program. It looks much too complicated and
>> spartan. IMO there are already much better tools for making music for the
>> C64, like GoatTracker or OdinTracker for example...

Heh, which is it: too complicated, or too spartan? :) Those must've been pretty informative screenshots to lead to such conclusions! :)

>I checked it out and i must agree. I noticed it is too far from the common
>'standards' of music player/editor. The user interface and keyboard logic
>is far from the common usage of most 'professional' kind of music editors
>around.

Well, all you've said is, "It's not what I'm used to." But the question is: what doesn't it do that you are looking for?

Yes, the user-interface is different. In fact, it's a bit experimental. The reason is simple: I don't think very highly of the "standard" interfaces, and am trying to explore other more effective ways of doing stuff. It's not 1990 anymore; why should I limit myself to yet another clone of a 10-15 year old music program/system?

The goal of the interface is to present useful information while keeping clutter (i.e. useless information) to a minimum. The system is also designed to let you compose "by ear", rather than by eye -- you compose by single-stepping through your notes, instead of staring at the screen -- and to do so rapidly.

As to the "keyboard logic", well, I don't know what to say. Musical keyboard layout, cursor keys to move around, and CTRL-letter keys for editor commands. It's awfully hard for me to see how this could be difficult or confusing.

Iseq is almost trivial to use: you pick an instrument, type a key, you hear the note as it is entered. There's nothing to set up -- you just type and run with it. Instruments take a few seconds to create, are easy to tweak and mess with until you find something you like, and can be simple or very complicated. Arpeggios too are trivial, and flexible. In summary, you can do pretty much anything you want, both musically and in terms of SID, and do so quickly and easily.

So where's the problem?

I figured when I wrote iseq that it wouldn't be very popular -- most sceners don't like to step out of the comfort zone. And it certainly has room for improvement, and I don't claim that it does everything or is for everyone. But it certainly does most things, does them well, and has capabilities that are sorely lacking in other music programs. With iseq, I can make pretty much

any sound I want, and write any type of music I want, and do it _quickly_ and easily, without fighting the player. I get to _play_, both with SID and with music -- I get to have FUN!

Who will use iseq? I will, of course!

>With first tests it hungs when adjusting the speed

I'm sorry, could you be more specific? (If there's a bug, I'll need more information).

>and the loader doesn't seem to work with Action Replay cartridge.

This is a strange thing, and the fault of AR. The "loader" is just the kernal LOAD routine -- totally standard. For some reason AR seems to get confused, maybe when the call to LOAD is done from bank 2.

> The tempo change to CIA
>timers sounds funny, but it makes a lot of restrictions in demo and game
>needs. Thought, it is said in the docs it can be played with VIC's
>rasterrupt too.

Heh. This always seems to baffle some people. Using the CIA timers _removes_ restrictions.

To play a tune, you call the play routine (JSR $1003) at fixed intervals. You can call the routine at the screen refresh rate; you can call it once every second. There's nothing that _forces_ you to use a CIA timer (that would be silly). Similarly, if you set the CIA timer to the frame rate, the tune will sound exactly the same when called from a VIC interrupt.

Some of us simply don't like being limited to some multiple of the frame rate -- not all music is for raster-timed demos and games. Iseq simply gives you the choice; the only place where CIA timers are necessary is doing true tempo changes. Otherwise, just use the default timer values (which are automatically set to the NTSC or PAL frame rate when you clear a tune), and it'll be fine.

Moreover, using the timers means that PAL and NTSC tempos are much closer -- a PAL guy can listen to an NTSC tune at pretty much the tempo the composer intented, without effort.

>I wouldn't recommend Iseq for any serious music making.

Heh. Well, once again, I have to ask: what can't you do in Iseq that is needed for "serious" music making? I'll bet the vast majority of these so-called serious tunes can be _easily_ duplicated in iseq.

Well, look; I know that Iseq isn't for everybody. Heck, aside from jpz I don't think anyone besides me uses any of my other programs, and I don't expect anything different for Iseq. But this "I spent five minutes with it/looked at the screenshots and it was unfamiliar, so it must be useless" stuff is a little silly. If you can say you gave it an honest try, that after understanding the system and working with it you see some weaknesses -- great! I'd love to hear about it!

After living and breathing (and playing) music my whole life, and writing
c64 music for a few years, I tried to write a music system which fixed
up the weaknesses of other programs and had some features I've always wanted.
Iseq has a logic and structure to it that is, yes, different than the norm,
but personally, I rather enjoy it. Why not give iseq an honest try?

cu,

-S

From:Blackspawn (noone@home.net)
subject:Re: Iseq music composition program released
Newsgroups:comp.sys.cbm, alt.c64
Date:2002-06-11 02:25:03 PST

Steve Judd schrieb:

>
> Heh, which is it: too complicated, or too spartan? :) Those must've been
> pretty informative screenshots to lead to such conclusions! :)

Complicated wasn't the right word perhaps. Impractical would be better. And it's
impractical because it's spartan. :)

> So where's the problem?

I just don't like it. That's all there is to say. When it comes to making music,
I'm am very used to the typical tracker interface, like Goat- oder OdinTracker
have (or FT2/IT2 etc on PC) and I could never imagine using a system like yours.
And I guess there will hardly be someone who can and will.

> With iseq, I can make pretty much
> any sound I want, and write any type of music I want, and do it _quickly_ and
> easily, without fighting the player. I get to _play_, both with SID and
> with music -- I get to have FUN!

Sounds to me like a typical tracker interface! Why didn't you implement one? It's
always good to have new ideas and invent new things, but not if they're just
useless and worse than the current standards.

> Who will use iseq? I will, of course!

I won't.

> Why not give iseq an honest try?

See above.

> cu,
> -S

/Blackspawn\

From:Anders Carlsson (anders.carlsson@mds.mdh.se)
Subject:Re: Iseq music composition program released
Newsgroups:comp.sys.cbm
Data:2002-06-11 02:59:42 PST

Blackspawn <noone@home.net> writes:

> And I guess there will hardly be someone who can and will.

I had a quick look at the Iseq page prior the release note, and thought
it looked funny, but also saw it was offered with some DOCUMENTATION.
Now, after these messages, I'm more than intrigued to download it and
give it a serious try. If it doesn't conform to the traditional tracker
standards doesn't matter as much as if it is powerful to work with and
able to produce the kind of music I'd like to compose.

Something that very many of the music editors for the C64 lacks is
well-written documentation. Not strange, as they often were hacks by
demo groups (and beware, any L4M3R who would use this 3L33T composer).
I find even the easiest of GUIs to include effects or details which
aren't self-explanatory. Admittenly, I haven't checked out Goat- or
OdinTracker in that context, but besides those, I only know one music
editor which comes with decent docs, and that is SIDwinder.

--
Anders Carlsson

From:Agemixer (agespam@NOSPAM.japo.fi)
Subject:Re: Iseq music composition program released
Newsgroups:comp.sys.cbm, alt.c64
Date:2002-06-11 02:51:30 PST

On 10 Jun 2002, Steve Judd wrote:

> Hola,
>
> In article <Pine.LNX.4.33.0206101321350.16410-100000@aapo.japo.fi>,
> Agemixer <agespam@NOSPAM.japo.fi> wrote:
> >
> >On Mon, 10 Jun 2002, Blackspawn wrote:
> >
> >> Robert Bernardo schrieb:
> >>
> >> > Steve Judd has released Iseq, a new music composition program for
> >> > the C64 (see Steve's description below). Program, examples,
> >> > docs, and screenshots are at www.ffd2.com/fridge/iseq
> >>I checked it out and i must agree. I noticed it is too far from the common
> >>'standards' of music player/editor. The user interface and keyboard logic
> >>is far from the common usage of most 'professional' kind of music editors
> >>around.
>
> Well, all you've said is, "It's not what I'm used to." But the question is:
> what doesn't it do that you are looking for?

Referring to the docs, it seems to be missing atleast some flexible wavelist/arpeggio, and seems to be replaced with 'fixf' feature. For example, how could i do a continuous bass sound sequence like this can be done in Iseq:?

```
Wave  trp+
 51    0c
 41    0c
 41    0c
 41(0) 00
 41(0) 00
 41(0) 00
 41(0) 00
 40    00
 40    00
 40    00
```

And an another issue, i didn't find a single word of "slide", "glide", "tie", or even "vibrato" effects from the docs, so i guess they are completely missing. Actually those are very important for the quality of the tune sounds and enjoyability... though most of those aren't made too well in some very known editors around, i had to replace the glitching glide sounds with my own methods with wavelists + ties, and even vibratos if the editor just made it possible.

> Yes, the user-interface is different. In fact, it's a bit experimental.
> The reason is simple: I don't think very highly of the "standard" interfaces,
> and am trying to explore other more effective ways of doing stuff. It's not
> 1990 anymore; why should I limit myself to yet another clone of a 10-15 year
> old music program/system?

Why do you think the old editors are worse? I have figured many old tools are a more like usable than nowaday's stuff in general..

I can't see a reason to make the user interface even harder than old-timer's editor :) Try to replace the CTRL+xx and SHIFT+xx keypresses with only one key press, like simply p = play (that megasound used to), or more common method that F1/F3/F5/F7 = play/stop/cont/f.forward (DMC-styled) or something similiar. The idea is to get rid of multiple keypresses for most common keypresses. But it is also a good idea to hide f.e. clear track, clear instruments undo commands and things like this, behind any press-together of either SHIFT, CBM or CTRL. For example, Home keypress goes to the first line, Clr to the end (or whaterver it does), but a less common CTRL+CLR clears the track and CBM+CLR could do the same. So you don't need to start to recall, if it was CTRL+CLR or CBM+CLR, but rather recall what this one key can do. I used to CTRL+del for UNDO command in my text editors.

It does not mean that the keyboard should not be a clone of an another C64 editor, but look at some old amiga/PC trackers, how similiar they were to use, comparing to C64 editors overall... If you are searching for some originality, i wouldn't do it with keyboard! :)

And why not implementing different keysets (DMC-like, JCH-like) to improve one's get-in-touch. Atleast I planned that for my tracker project years

ago.

> The goal of the interface is to present useful information while keeping
> clutter (i.e. useless information) to a minimum. The system is also
> designed to let you compose "by ear", rather than by eye -- you compose by
> single-stepping through your notes, instead of staring at the screen -- and
> to do so rapidly.

Sometimes it is good to have some visual information than no information
at all.

> As to the "keyboard logic", well, I don't know what to say. Musical keyboard
> layout, cursor keys to move around, and CTRL-letter keys for editor commands.
> It's awfully hard for me to see how this could be difficult or confusing.

I didn't say that it is hard to learn, but it could be more user friendly.

I remember the Voicetracker times, when you HAD to type the whole
C#4.1f command to get the note and a duration for it. Argh! :)

> I figured when I wrote iseq that it wouldn't be very popular -- most sceners
> don't like to step out of the comfort zone. And it certainly has room for
> improvement, and I don't claim that it does everything or is for everyone.

I guess the 'comfort zone' is not the problem, but if this editor was
simply better than the one you used to, you would like to dump your old
editor.

> >With first tests it hungs when adjusting the speed
>
> I'm sorry, could you be more specific? (If there's a bug, I'll need more
> information).

When you press cbm +/- some times (during playing) it reserves a lot of
rastertime and plays the tune in a high speed, but pressing any keys
has no effect.

> >and the loader doesn't seem to work with Action Replay cartridge.
>
> This is a strange thing, and the fault of AR. The "loader" is just the kernal
> LOAD routine -- totally standard. For some reason AR seems to get confused,
> maybe when the call to LOAD is done from bank 2.

Ok. (It may be an emulator bug too, sorry.)

> > The tempo change to CIA
> >timers sounds funny, but it makes a lot of restrictions in demo and game
> >needs. Thought, it is said in the docs it can be played with VIC's
> >rasterrupt too.
>
> Heh. This always seems to baffle some people. Using the CIA timers
> _removes_ restrictions.
>
> To play a tune, you call the play routine (JSR $1003) at fixed intervals.
> You can call the routine at the screen refresh rate; you can call it once

> every second. There's nothing that _forces_ you to use a CIA timer (that
> would be silly). Similarly, if you set the CIA timer to the frame rate,
> the tune will sound exactly the same when called from a VIC interrupt.

It _does_ the restrictions, if one made a tune with CIA timing, and i
need to use it with a fullscreen FLI demo or raster timing stuff :)

> Some of us simply don't like being limited to some multiple of the frame
> rate -- not all music is for raster-timed demos and games. Iseq simply gives
> you the choice; the only place where CIA timers are necessary is doing true
> tempo changes. Otherwise, just use the default timer values (which are
> automatically set to the NTSC or PAL frame rate when you clear a tune),
> and it'll be fine.
> >I wouldn't recommend Iseq for any serious music making.
>
> Heh. Well, once again, I have to ask: what can't you do in Iseq that is
> needed for "serious" music making? I'll bet the vast majority of these
> so-called serious tunes can be _easily_ duplicated in iseq.

Hmm, a so-called 100% cover of Commando tune by Rob Hubbard could do
fine :)

/hvsc/hvsids-4.6/Hubbard_Rob/Commando.sid
/hvsc/hvsids-4.6/VARIOUS/A-F/Agemixer/Commando_Remix.sid

which i covered with DMC 4.0B, this editor is easy but one of
hardest ones for covering anything well. :)

> Well, look; I know that Iseq isn't for everybody. Heck, aside from jpz I
> don't think anyone besides me uses any of my other programs, and I don't
> expect anything different for Iseq. But this "I spent five minutes with
> it/looked at the screenshots and it was unfamiliar, so it must be useless"
> stuff is a little silly. If you can say you gave it an honest try, that
> after understanding the system and working with it you see some weaknesses --
> great! I'd love to hear about it!

Most of the judgement is based on first experience and feeling.
But now you know atleast one who even bothered to test the editor =)

Those horizontally placed notes in sequences in iseq is very strange for
music. Not that the common oldie note and delay staves with G-key are
also... :)

It is also missing the on-screen octave information, that is important to
see the current playing note octave.

A some kind of sequence position is necessary for editing, too.

> After living and breathing (and playing) music my whole life, and writing
> c64 music for a few years, I tried to write a music system which fixed
> up the weaknesses of other programs and had some features I've always wanted.
> Iseq has a logic and structure to it that is, yes, different than the norm,
> but personally, I rather enjoy it. Why not give iseq an honest try?

I know the feeling, there is a lot of 'unfinished' editors around that

either weren't flexible enough, or, the music routines took a lot of rastertime... So i decided to start my own tracker project, meanwhile a lot of people wanted to kill my project "aren't there already enough of good trackers around yet", "This will become nothing anyway, everybody already uses their own music players", "This kind of editor is already being under construction, see this link blahblah.." and mumblings like that.

But still, i have not seen a single music editor project a 'complete' kind of tracker which can do it until now, with all the good features imagined and still optimized the final result for you without crashing.

My project become too large to handle with: 'flexible' user interface, 'flexible' player, 'variable speed' like iseq, even possible to stretch to be synchronized with 3-channels sound frequency to produce 'SLTS samples', free IRQ/NMI for possible sample addition, different editor sequence looks like tracker/DMC-style-expanded and that kind of 'crap' and cleverly optimizing tune packer. A little revolutionary, yeah, but It was a too large project and nobody seemed to like the idea of a new, a 'good-enough' tracker which has all the other editor's capabilities, with which you can torture your cat by playing him and even backwards... (?!)

Keep up your good work.

--
Agemixer/SCL

From:Steve Judd (sjudd@swcp.com)
Subject:Re: Iseq music composition program released
Newsgroups:comp.sys.cbm, alt.c64
Date:2002-06-17 22:15:21 PST

Hola Agemixer :)

In article <Pine.LNX.4.33.0206110957380.21613-100000@aapo.japo.fi>,
Agemixer <agespam@NOSPAM.japo.fi> wrote:
>
>On 10 Jun 2002, Steve Judd wrote:
>
>Referring to the docs, it seems to be missing atleast some flexible
>wavelist/arpeggio, and seems to be replaced with 'fixf' feature. For
>example, how could i do a continuous bass sound sequence like this can
>be done in Iseq:?
>
>Wave trp+
> 51 0c
> 41 0c
> 41 0c
> 41(0) 00
> 41(0) 00
> 41(0) 00
> 41(0) 00
> 40 00
> 40 00
> 40 00

Well, I'm not sure what the trp+ is, but I assume the "Wave" settings are $d404 settings. In Iseq, I'd just create a sequence of length 8 and enter those values (although I wouldn't use $51 :) -- when I used mixed waveforms in the past they didn't work on lots of SIDs).

Instead of a program with a set of built-in features, I've had this sort-of vision of a flexible set of tools that let you manipulate SID and compose music -- a "shell" with a set of musical building blocks that you could define and combine into anything you wanted, that let you control exactly what happens and give you total control over SID and the player's behavior (not a programming language; just a set of tools and a framework). That motivated a lot of Iseq.

Any music player can only write to SID every time it's called. In iseq, you specify exactly what gets written on each call. If you want to modify the frequency register, you do it; same for the filters, ASDR, waveforms, etc.; all registers are available to each instrument. The goal is to give you full control over what gets written to SID.

In addition to plain sequences -- which can be written continuously or just once -- there are also "modulators", for example ramps, which can go from A to B step C, and adders, which can add sequences of values.

The "fixf" thingie is something different. fixf just restores the frequency after a little bit -- for example, the bass+hihat example instrument sets the waveform to noise, then triangle, then pulse; meanwhile, the frequency needs to be high for the first two player calls, then restored to its original frequency. That is:

1st player call: waveform=$81 freq=$a000 (or whatever)
2nd player call: waveform=$11 freq=$a000
3rd player call: waveform=$41 freq=original value

fixf restores the frequency, that's all.

All of these are meant to be tools, for manipulating SID. They are actually compiled into the player during editing, and this makes it very easy to add more. Anything you want, to manipulate SID, can potentially be added in.

Arpeggios are in there -- ctrl-a from the instrument editor brings up the arpeggio editor. Arpeggios can be any length and speed, can have positive and negative offsets, and can either run continuously or one-shot.

>And an another issue, i didn't find a single word of "slide", "glide",
>"tie", or even "vibrato" effects from the docs, so i guess they are
>completely missing. Actually those are very important for the quality of
>the tune sounds and enjoyability... though most of those aren't
>made too well in some very known editors around, i had to replace the
>glitching glide sounds with my own methods with wavelists + ties, and even
>vibratos if the editor just made it possible.

Well, like a lot of things I probably was dumb and didn't mention it in the docs, but part of the Iseq philosophy of "you control the SID" is that it doesn't have built-in effects. Instead, it tries to provide the tools

you need to build your own effects. It tries to not limit you to whatever effect is built into the player.

So, for example, the instrument editor has a thingie called "frq+". With frq+, values are added to the current frequency. So if you want a vibrato, you enter a set of frq+ values to increase and decrease the frequency, and loop it continuously:

frq+ 0100 0200 0100 0000 ff00 fe00 ff00 0000

On the downside, it's not "proportional" -- you wouldn't use the same modulations at low and high frequencies. On the plus side, you get to specify exactly what you want, and combine it with stuff to make more complex effects; for example, it's easy to, say, combine a vibrato with a rising frequency, or make little ornaments for notes, or an 'asymmetric' vibrato -- anything you can think of.

This is also the idea for slides and glides -- again, the idea was that they're not built in, but the tools are there to make them. They can also be done using the arpeggio editor. Since this is kind-of a crappy way of implementing them, philosophical purity of not having any built-in fx may be abandoned in a future release :). But from your descriptions above, it sounds like building your own slides was necessary anyways. So the question is: is there a _better_ way of implementing slides, a way which lets you, the composer, control exactly what happens? I couldn't think of one, but I'm very open to ideas.

Ties are built into the player. Players typically do things automatically for the composer -- restart instruments, toggle gates, etc. With the aim of giving control to the composer, Iseq has switches for all those things. You can turn gate toggling on/off, restarts on/off, and so on. There are also two special notes, "hold" (continue previous note) and "rest" (zero frequency). One way to do a tie is to turn off instrument restarts and hold the note; another way is to just define a longer duration! Slurs and such are similar -- just turn off the parts that reset instruments.

It makes the player a little bulkier and less efficient, but again, I don't want the player to automatically assume what you want and do stuff for you -- I want you to have the abilitiy to control the behavior exactly.

>...
>more common method that F1/F3/F5/F7 = play/stop/cont/f.forward
>(DMC-styled) or something similiar. The idea is to get rid of multiple
>keypresses for most common keypresses. But it is also a good idea to hide
>f.e. clear track, clear instruments undo commands and things like this,
>behind any press-together of either SHIFT, CBM or CTRL. For example, Home
>...
>And why not implementing different keysets (DMC-like, JCH-like) to improve
>one's get-in-touch. Atleast I planned that for my tracker project years
>ago.

Good idea. Thanks for the suggestions/tips; I'll keep them in mind!

I tried to make common keypresses single-key -- notes, durations, etc. -- and group other functions together under the ctrl-keys. I just couldn't

stomach using something like "m" and "n" for "turn gate toggle off/on".
I think I would have ran out of keys anyways.

>I remember the Voicetracker times, when you HAD to type the whole
>C#4.1f command to get the note and a duration for it. Argh! :)

Yeah, that's what happened in my first player(s) (blahtune/tunesmith). It's not horrible, but it sure gets old fast. I think Iseq does a reasonable job of getting around this -- just changing durations when necessary -- but kind-of like a tracker I'm concerned that it gets me thinking about the composition in rigid intervals, i.e. every note becomes a 4/4 kind of note. (If that makes any sense :).

>> >With first tests it hungs when adjusting the speed
>>
>> I'm sorry, could you be more specific? (If there's a bug, I'll need more
>> information).
>
>When you press cbm +/- some times (during playing) it reserves a lot of
>rastertime and plays the tune in a high speed, but pressing any keys
>has no effect.

Hmmm. cbm+ doubles the playback speed, while cbm- halves it. If cbm+ is pressed too many times things will get out of control -- IRQs start happening faster than they can be acknowledged. But keys should still work. I'll have a look (and try to break it).

>> >and the loader doesn't seem to work with Action Replay cartridge.
>>
>> This is a strange thing, and the fault of AR. The "loader" is just the kernal
>> LOAD routine -- totally standard. For some reason AR seems to get confused,
>> maybe when the call to LOAD is done from bank 2.
>
>Ok. (It may be an emulator bug too, sorry.)

No, it's an AR thing, and you're absolutely right. I was stupid to not have a fix in place, since everyone and their brother uses AR. Programs that don't work with AR are programs that don't get run!

>> Using the CIA timers _removes_ restrictions.
>
>It _does_ the restrictions, if one made a tune with CIA timing, and i
>need to use it with a fullscreen FLI demo or raster timing stuff :)

Well, this is another place where I was stupid and left some important information out of the manual. Here's what I should have said: just set the first tempo value to $4cc7 (using ctrl-t) and it will behave just like a VIC interrupt.

$4cc7 is the PAL setting; $42c6 is the normal NTSC setting, and $417f is for old NTSC R56A VICs (64 cycles per line).

Also, when you clear a tune using shift-clear, the code checks whether the machine is PAL or NTSC and _should_ set the value appropriately -- another thing left out of the docs. Bleah.

Anyways, the bottom line is that you can make it work just like a normal VIC interrupt -- you're just not limited to a VIC interrupt.

>Hmm, a so-called 100% cover of Commando tune by Rob Hubbard could do
>fine :)
>
>/hvsc/hvsids-4.6/Hubbard_Rob/Commando.sid
>/hvsc/hvsids-4.6/VARIOUS/A-F/Agemixer/Commando_Remix.sid
>
>which i covered with DMC 4.0B, this editor is easy but one of
>hardest ones for covering anything well. :)

Heh, cool. OK, I'll take you up on it :). But you'll have to describe the instruments to me -- I'm pretty clueless as to duplicating instruments I haven't done before. (We can do this in email).

That is, if this is a test of my abilities -- well, I lose, I can do the notes from memory but I don't know how to make his instruments. But if it's a test of iseq's abilities, well, some information would be most helpful :).

>Most of the judgement is based on first experience and feeling.
>But now you know atleast one who even bothered to test the editor =)

Yep, fair enough, and I would surely have a similar reaction, if not worse; if I can't figure something out quickly I usually put the program aside, and I should have put more work into easing into this one. Oh well.

As to the second, what makes it REALLY rare is that you not only tried the program, but made COMMENTS on it! :) (That _never_ happens!)

>Those horizontally placed notes in sequences in iseq is very strange for
>music. Not that the common oldie note and delay staves with G-key are
>also... :)

Well, that's a funny thing. We read books left to right, and sheet music left to right. There's more columns than rows on the screen, too. So I began to wonder why 64 tunes are written like Chinese script.

One reason is of course that if the information is "wide", more of it fits in columns. And I learned another reason -- vertical stacking is a lot easier to code than horizontal (especially with variable-width fields)! But Iseq fits, what, 12 "tracks" on the screen at a time by using rows, and more notes are visible at a time. You can see at a glance which are used, which are left, which parts go where. It's taken me some getting used to, too, but, well, I actually like it!

>It is also missing the on-screen octave information, that is important to
>see the current playing note octave.

Well.... this is another one that got to bugging me. Tunesmith displays note, octave, and duration on the screen, and I realized that I never found the octave/duration information useful, but instead spent all my time single-stepping to find my place in the tune. Iseq displays the current octave at the top of the screen, and displays the octave in the "long" description of the note under the cursor. I did try putting the

octave information in with the note, but decided that

A B C# B C# A D

was a lot cleaner looking than

A-4 B-4 C#-4 B-4 C#-4 oops ran out of screen room

so it went into the long description instead (at the bottom of the screen).

>A some kind of sequence position is necessary for editing, too.

Hey, great idea :).

>I know the feeling, there is a lot of 'unfinished' editors around that
>either weren't flexible enough, or, the music routines took a lot of
>rastertime... So i decided to start my own tracker project, meanwhile a
>lot of people wanted to kill my project "aren't there already enough of
>good trackers around yet", "This will become nothing anyway, everybody
>already uses their own music players", "This kind of editor is already
>being under construction, see this link blahblah.." and mumblings like
>that.

Bah. That's Bull. I totally think you should write your own player. As you say, current programs do _not_ do everything they can or should do. Besides, you're an experienced and high quality composer, so why should you have to depend on others for your composition needs, especially when those needs aren't being met?

Now, if it means you'll have to stop composing for a while and/or get bogged down in a bunch of coding, well, forget it -- talk some coder info adding the features you want :). Or maybe someone has released some source code that can be modified. Otherwise, there's always room for more, new programs. That's what keeps the C= going, and if it makes you a more productive composer then where's the problem?

(I almost hate to mention it, but you can always tell ME what features you'd like to see :).

Anyways, I really do appreciate the comments, and the chance to defend poor little iseq (actually, it's something like 70 blocks of code -- not tables, graphics, music, and code; just code -- which is really humongous).

Further comments, good or bad, are unexpected but always welcome :).
I rarely read comp.sys.cbm though, so maybe email is better.

And Agemixer: get busy on the Ultimate Tracker! :)

cu all,

-Steve

'Commando' tune in Iseq

As an example of Iseq music composition, this is the Steve Judd cover of Rob Hubbard's Commando tune. The style is as in Iseq, with the hidden numbers in the editor added here for more reliability.

Voice tracks:

```
V0: hard+ gate=3 DDDDG trans=3 G trans=0 HIJIJKKLLMM trans=2 M trans=0 NPPPPTULL hard+ gate=3 MM
    trans=2 M trans=0 NPPPX

V1: hard+ gate=3 EEEEEE trans=3 EE trans=0 EEEEEEDD trans=-3 D trans=4 D trans=-3 D trans=4 D trans=5
    DD trans=7 DD trans=3 PPPP trans=0 DDDDDD trans=-3 D trans=4 D trans=-3 D trans=4 D trans=5 DD
    trans=7 DD trans=3 PPP trans=0 X

V2: hard+ gate=0 AAA trans=3 A trans=0 BAAC trans=5 C trans=-3 C trans=4 C trans=-3 C trans=4 C
    trans=5 CC trans=-5 CC trans=0 gate=3 QQQQ gate=0 C trans=5 C trans=0 C trans=5 C trans=0 C
    trans=5 C trans=-3 C trans=4 C trans=-3 C trans=4 C trans=5 CC trans=-5 CCQQQ trans=0 W
```

Patterns:

```
A: inst=0(hubbass) dur=1(6) a1 h a1 a2 inst=1(crash) a1 h inst=0(hubbass) a1 h a1 h h a2
   inst=1(crash) a1 h inst=0(hubbass) g2 a2 a1 h a1 a2 inst=1(crash) a1 h inst=0(hubbass) a1 h a1 h h
   a2 inst=1(crash) a1 h instr=0(hubbass) g2 a2 a#1 h a#1 a#2 instr=1(crash) a#1 h instr=0(hubbass)
   a#1 h a#1 h h a#2 instr=1(crash) a#1 h instr=0(hubbass) g#2 a#2 e1 h e1 e2 instr=1(crash) e1 h
   instr=0(hubbass) e1 h e1 h h e2 inst=1(crash) e1 h instr=0(hubbass) d2 e2

B: a1 h a1 a2 instr=1(crash) a1 h instr=0(hubbass) a1 h a1 h h a2 inst=1(crash) a1 h instr=0(hubbass)
   g2 a2 d2 h d2 d3 instr=1(crash) d2 h instr=0(hubbass) d2 h d2 h h d3 instr=1(crash) d2 h
   instr=0(hubbass) c3 d3 e1 h e1 e2 instr=1(crash) e1 h instr=0(hubbass) e1 h e1 h h e2
   instr=1(crash) e1 h instr=0(hubbass) d2 e2 a1 h a1 a2 instr=1(crash) a1 h instr=0(hubbass) a1 h a1
   h h a2 instr=1(crash) a1 h instr=0(hubbass) g2 a2

C: instr=0(hubbass) a1 h a1 a2 instr=1(crash) a1 h instr=0(hubbass) a1 h a1 h h a2 instr=1(crash) a1
   h instr=0(hubbass) g2 a2 a1 h a1 a2 instr=1(crash) a1 h instr=0(hubbass) a1 h a1 h h a2
   instr=1(crash) a1 h instr=0(hubbass) g2 a2

D: gate+ instr=2(crish-v1) dur=3(12) arp=1(octave) e6 a6 instr=4(hubharmo) dur=1(6) a4 h a4 h a4 h h
   a4 h h g4 h h a4 a4 h a4 h g4 h a4 g4 a4 h instr=2(crish-v1) dur=3(12) e6 a6

E: gate+ instr=3(crish-v2) arp=0 dur=1(6) b6 c7 c6 c6 instr=4(hubharmo) arp=1(octave) e4 h e4 h f4 h
   h e4 h h d4 h instr=3(crish-v2) dur=1(6) b6 c7 c6 c6 instr=4(hubharmo) e4 h e4 h e4 h h h h h h h

G: arp=0 instr=5(hublead) dur=1(6) rest rest a4 a4 a4 h a4 h a4 h h a4 h h a4 h h e5 e5 h e5 h e5 h
   e5 h h h instr=6(beoop) e3 h h h instr=5(hublead) f5 h h h e5 h h h f5 h h h e5 h d5 h h b4 b4 h
   b4 h b h b4 h h h instr=6(beoop) b3 h h h

H: inst=5(hublead) a4 a4 a4 h a4 h a4 h a4 h h b4 h h c5 h d5 d5 d5 h d5 h e5 h d5 h h h
   inst=6(beoop) c4 h inst=5(hublead) d5 e5 f5 f5 e5 h d5 h c5 h b4 h a4 h g#4 h h h h
   inst=5(hublead) a4 a4 h a4 h b4 h a4 h h h inst=6(beoop) c4 h h h

I: inst=4(hubharmo) arp=1(octave) dur=1(6) c5 b4 h a#4 a4 h c5 b4 h a#4 a4 h c5 b4 h a#4 a4 h c5 b4 h
   a#4 a4 h c5 b4 h a4 f5 h e5 h

J: f5 e5 h d#5 d5 h f5 e5 h d#5 d5 h f5 e5 h d5 b4 a#4 h a4 g#4 h b4 a#4 h a4 g#4 h c5 h b4 h

K: inst=5(bublead) arp=0 c5 b4 h a#4 a4 h c5 b4 h a#4 a4 h c5 b4 h a#4 a4 h c5 b4 h a#4 a4 c5 b4 h a4
   c5 h b4 h

L: gate- dur=3(12) inst=5(hublead) f#5 inst=a(scream-) f#5 h h inst=5(hublead) e5 h b4 a#4 h h h
   dur=1(6) a#4 h b4 a#4 b4 h c#5 h c#5 h c#5 h h h h h c#5 h e5 h h c#5 h h inst=b(scream+) b4 h h
   inst=5(hublead) c#5 h h inst=a(scream-) c#5 h gate+ dur=3(12) h h h inst=6(beoop) c6 c6

M: gate+ inst=5(hublead) dur=1(6) d5 h h h h h d5 h e5 h h d5 h h inst=b(scream+) c5 h h h
   inst=5(hublead) d5 h d5 h c5 h d5 c5 d5 h h h h h
```

```
N: e5 h h h h h e5 h f#5 h h e5 h h d5 h e5 h h f#5 h h g#5 h f#5 h h g#5 h h a5 h

P: inst=9(harmo2) arp=1(octave) dur=1(6) g5 g5 g5 f5 g5 h f5 g5 h f5 g5 f5 g5 h f5 h

Q: inst=9(harmo2) arp=1(octave) d#3 d#3 d#3 c#3 inst=6(beoop) arp=0 d#3 h inst=4(hubbarmo)
   arp=1(octave) c#3 d#3 h c#3 d#3 c#3 arp=0 inst=6(beoop) d#3 h c#3 h

S: gate- dur=3(12) inst=c(scream) g4 h inst=b(scream+) g4 h inst=c(scream) a4 h h h h g4 a4 d5 c5
   a4 h c5 h inst=b(scream+) c5 h inst=c(scream) d5 h h h c5 d5 g5 f#5 d5 a4 g5

T: arp=0 inst=c(scream) dur=1(6) a5 a5 h g5 a5 h g5 a5 h g5 f#5 g5 f#5 e5 f#5 e5 d5 d5 h c5 d5 h c5
   b4 h c5 b4 a4 b4 c5 d5 g5 a5 a5 h g5 a5 h g5 a5 h g5 f#5 g5 f#5 e5 f#5 e5 d5 d5 h c5 d5 h c5 b4 h
   c5 b4 a4 b4 c5 d5 g5

U: gate- dur=3(12) b5 inst=a(scream) b5 h h inst=c(scream) a5 h h h h h dur=1(6) g5 gate- h a5 h c6
   c6 a5 h g5 h a5 h dur=3(12) d6 inst=a(scream-) d6 h h inst=c(scream) c6 h h h h c6 d6 g6 gate+
   dur=1(6) f#6 f#6 dur=3(12) d6 c6 a5 gate- g5 inst=b(scream+) g5 h h inst=c(scream) a5 h h h h h
   dur=1(6) g5 h a5 h gate+ c6 c6 a5 h g5 h gate- a5 h dur=3(12) d6 inst=a(scream-) d6 h h
   inst=c(scream) c6 h h h h c6 d6 gate+ g6 dur=1(6) f#6 f#6 dur=3(12) d6 c6 a5

W: arp=0 dur=3(12) h h h h h h inst=6(beop) g6 g6

X: arp=0 dur=3(12) h h h h h gate- h inst=1(crash) e1 e1
```

Arpeggio:

```
0 <no arp>
1 [+0 +12]
```

Instruments:

inst 0-hubbass atdk=09 surl=d0 pwid=0102 01ff 0016 creg=41 gate=40	inst 1-crash atdk=0a surl=09 freq=0f82 1f04 creg=81 80 gate=88	inst 2-crish-v1 atdk=0d surl=f1 creg=15 81 81 15 gate=14	inst 3=cridh-v2 atdk=0f surl=b4 pwid=0200 creg=43 81 81 43 gate=42
inst=4-hubbarmo atdk=06 surl=ea pwid=01800 creg=41 81 81 41 gate=40	inst=5-bublead atdk=29 surl=99 pwid=0900 09a8 0a50 0af8 0ba0 0c48 creg=41 frq+=0000 0040 0080 00c0 00c0 0080 gate=40	inst=6-beop atdk=0a surl=0800 freq=0dd0 0100 0100 creg=41 81 81 40 gate=40	inst=9-harmo2 atdk=06 surl=c9 pwid=0800 0006 creg=41 81 81 40
inst=a-scream- atdk=38 surl=9a pwid=0b20 0c00 0ce0 0dc0 0ea0 0dc0 creg=41 gate=40	inst=b-scream+ atdk=29 surl=99 pwid=0900 09c0 0a80 0c00 0cc0 creg=41 frq+=0000 0028 gate=40	inst=scream atdk=38 surl=7a pwid=0b20 0c00 0ce0 0dc0 0ea0 0dc0 creg=41 frq+=0000 0094 0094 0094 0000 ff6c	inst f-hard

I very hope that this transposition give you an idea of how Iseq works and so you take the player and made some composition with it.

Ripping a game: DIG DUG

Stefano Tognon <ice00@libero.it>

In early months of 2001 I saw a Sid Hunt (http://lala.c64.org/sid-hunt) request for ripping the game "Dig Dug".

I was surprise to see that this game was not in HVSC (http://hvsc.c64.org) , because I consider it one of the best old game. So I decide to rip it by myself.

There are various technique for ripping a tune from a game or demo but I always use the most difficult and longer: disassemble the game and give a source for the rip.

However a brief guide description for ripping can be found at: http://www.geocities.com/SiliconValley/Lakes/5147/sidplay/docs.html#ripping

The reason for why I prefer this way is simple: I always like to know how the sound is physically made by the music author and looking in so deep level is a good way to understand that.

This technique is however probably the most indicate when ripping old games, where the music routines were together with the games routines.

Step 1: Get the game

The first step is to retrieve a image file of the game by looking in the common games archive in the net. It is very important to download all the images you found, because:

1) There are various cracked version of the game and sometimes the cracker had into duce some bugs in the music data during his work, so you must look for the file that seems the most accurate.
2) You should use the image that come with no intro o similar matter, because the intro can contains music and so when you look for it in the game, you could saw that part in memory.
3) You should use the image file that start directly the game (like a .prg not compressed), so you can use a disassembly directly to the file instead of from a memory snapshot.

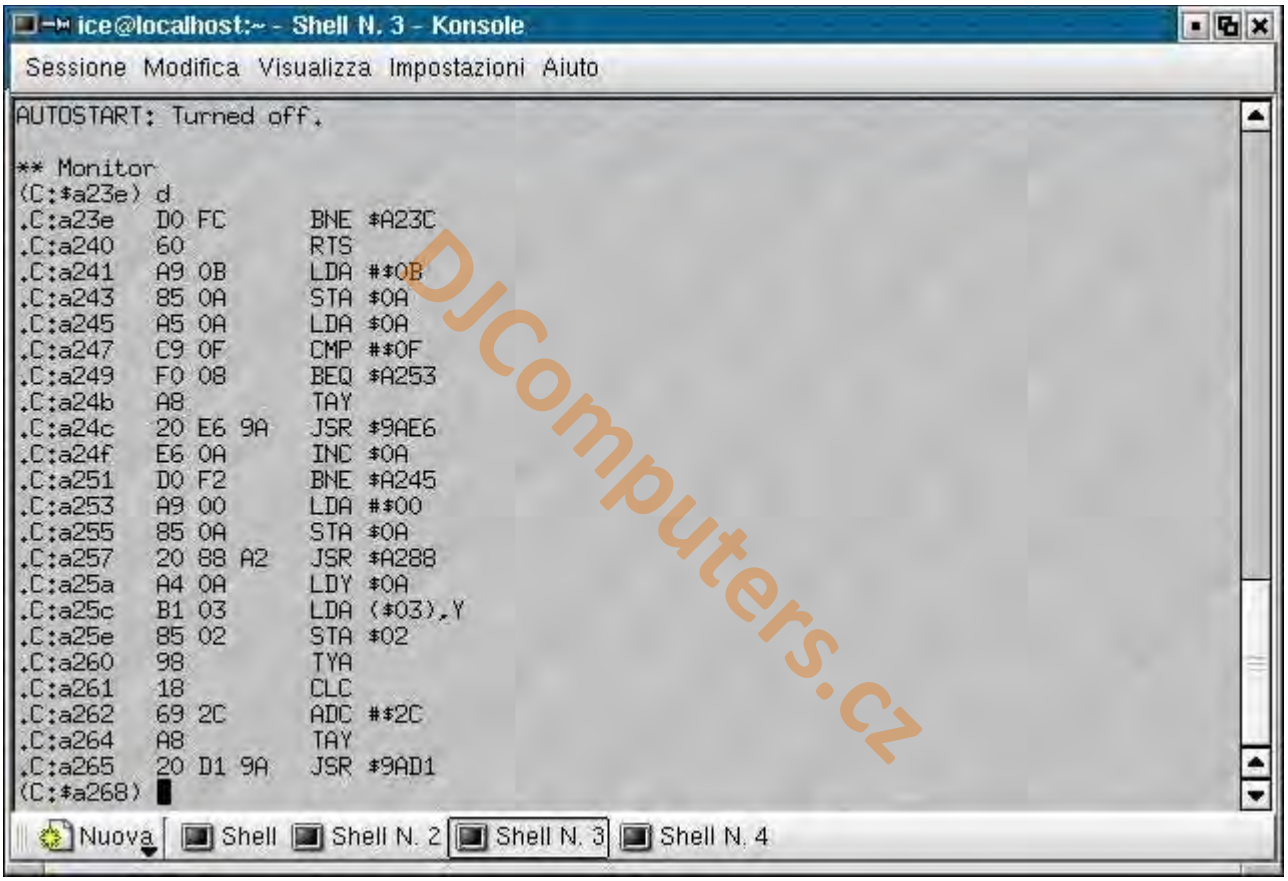
For *Dig Dug* I have use the cartridge image of the game, because all the game is within the image without extra stuff.

Step 2: Play the game!

Even if these seems a stupid observation, it can be very useful to know when and where the music is played in the game. When you are disassembly the game you could figure better where the music routine is. You should use even a monitor (like that come in Vice emulator) for keeping a list of addresses that are executed during the game by setting some breakpoints..

Step 3: Disassemble the game

There are various mode to obtain a disassemble of the game. You should use the Vice monitor for saving the disassemble:



However, you should even dump the memory of the game with the monitor in a file and then disassemble it with a more specific tool.

For example you can use the *sid_dis* tools that produce a output like that of Vice's monitor from a sid file. In this case you should create a fake sid file that contains the image you want to disassemble (the tool is available from here: http://www.geocities.com/SiliconValley/Lakes/5147/packages/).

You probably know other tools for disassemble a file: I use the myself utility: *jc64dis* (http://sf.net/projects/jc64) that disassemble prg, sid and mus file. One characteristic of this tool is to add a comment to the known memory locations being disassembled.

This can be a good point for two reasons:

1) It is more simple to look for specific memory locations while try to understand the code you are being analyse. For example you may look for the string"0314" when looking for a IRQ setting, but if you simple look for"IRQ" you probably found more stuff related to the IRQ like a JSR $EA31 call.
2) As I would produce a source, it is more beautiful to have a commented code.

In the specific case of Dig Dug I had save the memory location from $8000 to $BFFF (the cartridge image) from the Vice's monitor (with the *save* command) and that give the prg resulted file to *jc64dis*.

Step 4: Let the order emerge from the chaos!

If you look at the jc64dis's listing, you will see that there are about 8300 rows of disassembled code! At the first instance is very important to give a look to the *lst* file in all his extension for viewing where there is the program code and where there is the data (you should see lot of NOOP or JAM instructions here).

As we would found the music code, we should search for string like"Voice", "Freq", "Control", etc. in the listing. When we found this, we will evidence the code by adding new lines in it and remove the line number of not used locations like in this examples:


```
BBFE  A6 EA      LDX  $EA        Table of screen line/Transient editor
BC00  95 E1      STA  $E1,X      Table of screen line/Transient editor
BC02  94 E0      STY  $E0,X      Table of screen line/Transient editor
BC04  4C F3 BA   JMP  $BAF3      BASIC ROM
BC07  A9 11      LDA  #$11
BC09  D0 0A      BNE  $BC15      BASIC ROM
BC0B  A9 10      LDA  #$10
BC0D  D0 06      BNE  $BC15      BASIC ROM
BC0F  A9 41      LDA  #$41
BC11  D0 02      BNE  $BC15      BASIC ROM
BC13  A9 40      LDA  #$40
BC15  A4 F6      LDY  $F6        Vector: keyboard decode table
BC17  99 04 D4   STA  $D404,Y    Voice 1: Control registers
BC1A  4C F3 BA   JMP  $BAF3      BASIC ROM
BC1D  A9 02      LDA  #$02
BC1F  D0 02      BNE  $BC23      BASIC ROM
BC21  A9 06      LDA  #$06
BC23  A4 EB      LDY  $EB        Table of screen line/Transient editor
BC25  99 F2 00   STA  $00F2,Y    Label of screen lines
BC28  A9 40      LDA  #$40
BC2A  85 FE      STA  $FE        Free 0 page for user program
BC2C  4C F3 BA   JMP  $BAF3      BASIC ROM
```

That will become like in this peace of code:

```
BBFE  A6 EA      LDX  $EA         Table of screen line/Transient editor
BC00  95 E1      STA  $E1,X       Table of screen line/Transient editor
BC02  94 E0      STY  $E0,X       Table of screen line/Transient editor
BC04  4C F3 BA   JMP  $BAF3       BASIC ROM


BC07  LDA  #$11
      BNE  $BC15
      LDA  #$10
      BNE  $BC15
      LDA  #$41
      BNE  $BC15
      LDA  #$40
BC15  LDY  $F6          ;Vector: keyboard decode table
      STA  $D404,Y      ;Voice 1: Control registers
BC1A  JMP  $BAF3


BC1D  A9 02      LDA  #$02
BC1F  D0 02      BNE  $BC23       BASIC ROM
BC21  A9 06      LDA  #$06
BC23  A4 EB      LDY  $EB         Table of screen line/Transient editor
BC25  99 F2 00   STA  $00F2,Y     Label of screen lines
BC28  A9 40      LDA  #$40
BC2A  85 FE      STA  $FE         Free 0 page for user program
BC2C  4C F3 BA   JMP  $BAF3       BASIC ROM
```

As you can see, only the part that contains music code is changed, removing the line number and data, leaving only the code, the comments and the used memory locations, like BC07 (start of routine), BC15 (a local reference) and BC1A (end of routine).

After this task, a good point is to find where are the irq interrupts like NMI or IRQ used in the game. You may so look for code that use locations $0314-0319:

```
9DBE  A9 FE      LDA  #$FE
9DC0  8D 00 DC   STA  $DC00       Data port A #1: keyboard, joystick, paddle,
                                  optical pencil
9DC3  A9 E1      LDA  #$E1
9DC5  8D 14 03   STA  $0314       Vector: Hardware Interrupt (IRQ)
9DC8  A9 9D      LDA  #$9D
9DCA  8D 15 03   STA  $0315       Vector: Hardware Interrupt (IRQ)
9DCD  A9 20      LDA  #$20
9DCF  8D 12 D0   STA  $D012       Reading/Writing IRQ balance value
9DD2  A9 01      LDA  #$01
9DD4  8D 19 D0   STA  $D019       Interrupt indicator register
9DD7  8D 1A D0   STA  $D01A       IRQ mask register
9DDA  AD 0D DC   LDA  $DC0D       Interrupt control register CIA #1
9DDD  58         CLI
9DDE  4C 12 9E   JMP  $9E12       Normal space for BASIC programs/ROM cartridge
```

So we find that the game set the IRQ at $9DE1 and that the IRQ is triggered by the VIC raster routine $20. It is very important to see all the interrupts generator, because some games uses both CIA and VIC IRQ interrupt and even NMI interrupt.

One way to be sure witch interrupt the music routine use, is to stop one of the IRQ source (e.g CIA or VIC) by modify some memory locations with the emulator monitor. If you still listen the music then the IRQ source you have stopped is not the one responsible for it.

However, as soon as the IRQ start routine is fount, we can examine it for seeing when it goes to call the routines related to the music, and then follow the code flow to figure all the music player.

As we have looked at all the music routines, we may found the data used by those routines. For examples, frequency could be taken from a table, and as the SID register are not readable, there were some locale storage for some variables related to each voices. Else, notes can be taken from patterns and tracks...

For examples, after a good look inside the listing, we could figure out that at $BA0E there is a routine for setting the track currently used, because it go to manage two memory locations as pointer to memory data. Each of this track data contains meta instructions used by the player to generate the music, as we can see by looking at the data around $Bcxx-BDxx.

As soon as we found some memory location data, it is good to manage them in a better manner, by adding .byte instructions like this:

```
track9:
  .byte $38, $0F                ; set volume to 15
  .byte $20, $07, $00           ; set ADSL
  .byte $58, $10                ; set filter mode
  .byte $28, $00, $87           ; set filter frequency
  .byte $30, $28                ; set resonance
  .byte $78, $02                ; set number of repeat (A)
BD60
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $3C, $32           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $70, <rBD60, >rBD60     ; goto repeat (A)
```

Step 5: Make a source file

Now, I suppose that all the code related to the music is isolated and converted with the syntax of the previous examples. Now it is simple: remove all the not modified code lines like

```
AC5A  A5 57      LDA  $57         Scratch for numeric operation
AC5C  D0 01      BNE  $AC5F       BASIC ROM
AC5E  60         RTS
```

from the code like (this is a part that set a track):

```
BA0E
      sta  $EB
      asl  a
      sta  $EA
```

Now, give useful comments to each routines and then change each memory references with a more human readable name like:

```
;================================
; set the tracks and voice for it
;================================
setTrack:
      sta  $EB
      asl  a
      sta  $EA
```

At this point the code is in a form that a cross compiler can understand: there is only to setting up some initialization part in the start of the code. For example I use to add both code for generating a prg files runnable from a C64 emulator and a psid files runnable from a sid player.

Step 6: listen to the compiled source

Now, compile the source and listen to the result with as many sidplayer/emulators you know or preferably with the real C64.
This is for being sure that the rip sounds as in the real stuff and so you have done a good work.

Inside DIG-DUG

The complete source of the rip is placed in the next paragraph, but now it's time to analyse how the sound is generated in the game.

As you can see from the game, the sound is changed according to the action that you are done with your player: at my point of view this made the simple sound of Dig Dug more interesting.

In the game there are 8 pieces of music: the routine called *setTrack* use the passed parameter for setting the music track to use for each voices. Each track body is composed by a sequence of bytes that are used as meta instructions, like:

1) set the volume
2) set ADSR
3) set control registers
4) set frequency
5) set note length duration
6) set amplitude
7) set filter mode
8) set filter frequency
9) set resonance

and other instructions for allow a flow control, like:
1) set the number of repeats to execute
2) goto a specific part

The instructions are decoded by a table that points directly to the music routines.

So, a typical note pattern looks like:

```
.byte $80                       ; set control to 41 (rectangle)
.byte $08, $18, $0E             ; set frequency
.byte $98                       ; set note length duration to 6
.byte $88                       ; set control to 40 (rectangle no A)
.byte $90                       ; set note length duration to 2
```

As you see the gate bit is released for some time, but there are parts that never clears the gate bit (using short note lengths instead).

Even if this kind of player can seems so different from today music player, it is however modular and the code is no so nested with the game code as I expect to found from a so old game.

The Code:

A note: this code is a my reverse engineering of an existent work that is copyrighted by the respective author/software house. As at the moment in Italy is allowed to made a reverse engineering, it is not clean if this can be distributed in a work like this. So you are not allowed to do any work to this listing that can damage the copyright owner!

```
; Dig Dug game
; copyright 1982 Namco, 1983  Atari, 1984 Datasoft
; The music in the game are changed dinamically
; during the games

.ifdef sid
 .byte "PSID"
 .word $0200          ; version 2
 .word $7C00          ; data offset
 .word $0000          ; load address in cbm format
 .byte >initmusic
 .byte <initmusic
 .byte >play
 .byte <play
 .word $0800          ; 8 song
 .word $0700          ; default song 7
 .word $0000
 .word $0000
 .byte "Dig Dug",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .byte "<?>",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .byte "1984 Datasoft",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .word $0000
 .word $0000
 .word $0000

.endif

.byte $01,$08
.byte $0b,$08,$e8,$03,$9e,"2061",0,0,0
.org 2061
 jsr initmusic

 sei
 lda #<interr
 sta $0314
 lda #>interr
 sta $0315
      ldx  #$00
      stx  $DC0E       ; Control register A of CIA #1
      inx
      stx  $D01A       ; IRQ mask register
 cli

aaa:
 jmp aaa

interr =*
.ifndef sid
     lda  #$01
      sta  $D019       ; Interrupt indicator register
      lda  #$82
      sta  $D012       ; Reading/Writing IRQ balance value
      lda  #$1B
      sta  $D011       ; VIC control register
      lda  #$01
      sta  $D020       ; Border color
.endif

 jsr play

.ifndef sid
    dec  $D020       ; Border color
.endif

 jmp $ea31

;===================================
```

Conditional code for the sid file header

```
;init music
initmusic:

.ifndef sid
  lda  #$06
.endif

  cmp #$00
  beq t1

  cmp #$01
  beq t2

  cmp #$02
  beq t3

  cmp #$03
  beq t4

  cmp #$04
  beq t5

  cmp #$05
  beq t6

  cmp #$06
  beq t7

  lda #$0a
  jsr setTrack
  lda #$0c
  jsr setTrack
  lda #$0d
  jsr setTrack
  rts

t1:
  lda #$00
  jsr setTrack
  lda #$01
  jsr setTrack
  lda #$02
  jsr setTrack
  rts

t2:
  lda #$0b
  jsr setTrack
  lda #$0c
  jsr setTrack
  lda #$03
  jsr setTrack
  rts

t3:
  lda #$0b
  jsr setTrack
  lda #$0c
  jsr setTrack
  lda #$04
  jsr setTrack
  rts

t4:
  lda #$0b
  jsr setTrack
  lda #$0c
  jsr setTrack
  lda #$05
  jsr setTrack
  rts

t5:
  lda #$0b
  jsr setTrack
  lda #$0c
  jsr setTrack
  lda #$06
```

Select the subtune to play according with the passed A register

Init the tracks for first tune to play

```
  jsr setTrack
  rts

t6:
  lda #$0b
  jsr setTrack
  lda #$0c
  jsr setTrack
  lda #$07
  jsr setTrack
  rts

t7:
  lda #$08
  jsr setTrack
  lda #$09
  jsr setTrack
  lda #$0d
  jsr setTrack
  rts

;================================
; set the tracks and voice for it
;================================
setTrack:
      sta  $EB
      asl  a
      sta  $EA
      tya
      pha
      txa
      pha
      ldy  $EB
      lda  voices,y             ; read voice number
      sta  $EB                  ; voice number
      asl  a
      tax
      ldy  $EA
      sta  $EA
      lda  tracks,y             ; read instructions track
      sta  $E0,x
      sta  $EC,x
      lda  tracks+1,y
      sta  $E1,x
      sta  $ED,x
      ldx  #$02
loopST:
      lda  $F2,x
      clc
      adc  $F5
      sta  $F2,x
      dex
      bpl  loopST
      ldx  $EB
      lda  #$00
      sta  $F2,x
      lda  #$01
      sta  $F5
      pla
      tax
      pla
      tay
      rts

voices:
  .byte $00, $01, $02, $02
  .byte $02, $02, $02, $02
  .byte $00, $01, $00, $00
  .byte $01, $02

tracks:
  .byte <track1, >track1
  .byte <track2, >track2
  .byte <track3, >track3
  .byte <track4, >track4
  .byte <track5, >track5
  .byte <track6, >track6
  .byte <track7, >track7
```

```
  .byte <track8, >track8
  .byte <track9, >track9
  .byte <track10, >track10
  .byte <track11, >track11
  .byte <track12, >track12
  .byte <track12, >track12
  .byte <track12, >track12

play:
      dec  $F5
      beq  locA
      rts
locA:
      lda  $D0
      beq  locB
      rts
locB:
      lda  #$01
      sta  $D0
      jsr  locC
      lda  #$00
      sta  $D0
      rts
locC:
      cli
      lda  $DC0D                ; Interrupt control register CIA #1
      ldy  #$02                 ; last voice
      ldx  #$04
nextVoice:
      lda  $E1,x                ; is active?
      beq  noActive
      lda  $00F2,y              ; note length duration
      bne  noActive
      lda  voiceIndex,y
      sta  $F6                  ; voice index
      jsr  playNext
noActive:
      dex
      dex
      dey                       ; previous voice
      bpl  nextVoice
      jmp  jmpAdjustDuration

voiceIndex:
  .byte $00, $07, $0E           ; index of the 3 voices

;==================================
; play next note in track
;==================================
playNext:
      txa
      pha
      tya
      pha
      asl  a
      stx  $EA
      sty  $EB
nextInstr:
      ldx  $EA
      ldy  $EB
      lda  ($E0,x)              ; read actual track instruction
      sta  $FE                  ; store readed value
      cmp  #$B1
      bcs  skipExecute
      and  #$07
      bne  skipExecute
      lda  $FE                  ; read stored track instruction
      cmp  #$B0                 ; end?
      bne  executeInstr
skipExecute:
      lda  #$00
      sta  $E1,x                ; no active
      lda  #$7F
      sta  $00F2,y              ; note length duration
      jmp  exitPlay

executeInstr:
      jsr  incForNext           ; increment pointer for next reading
```

```
      lda  $FE                  ; read stored track instruction
      and  #$F8
      lsr  a
      lsr  a
      tay
      lda  instTable,y          ; read this instruction from table
      sta  $E6
      lda  instTable+1,y
      sta  $E7
      jmp  ($00E6)              ; execute the readed instruction
finishInstr:
      lda  $FE                  ; read stored track instruction
      cmp  #$40                 ; is set note length duration
      bne  nextInstr            ; exit only when note is set
exitPlay:
      pla
      tay
      pla
      tax
      rts

jmpAdjustDuration:
     jmp  adjustDuration

;==================================
; increment pointer for next reading
;==================================
incForNext:
      lda  $E0,x
      clc
      adc  #$01
      sta  $E0,x
      lda  $E1,x
      adc  #$00
      sta  $E1,x
      rts

;==================================
; set frequency control
;==================================
setFrequency:
      jsr  readNext             ; read next song instruction/data
      ldy  $F6                  ; voice index
      sta  $D400,y              ; Voice 1: Frequency control (lo byte)
      jsr  readNext             ; read next song instruction/data
      sta  $D401,y              ; Voice 1: Frequency control (hi byte)
      ldx  $EB                  ; voice number
      sta  $D8,x
      jmp  finishInstr

;==================================
; set voice control
;==================================
setControl:
      jsr  readNext             ; read next song instruction/data
      ldy  $F6                  ; voice index
      sta  $D404,y              ; Voice 1: Control registers
      jmp  finishInstr

;==================================
; set wave form pulsation amplitude
;==================================
setAmplitude:
      jsr  readNext             ; read next song instruction/data
      ldy  $F6                  ; voice index
      sta  $D402,y              ; Voice 1: Wave form pulsation amplitude (lo byte)
      jsr  readNext             ; read next song instruction/data
      sta  $D403,y              ; Voice 1: Wave form pulsation amplitude (hi byte)
      jmp  finishInstr

;==================================
; set the Attack/Decay Sustain/Release
;==================================
setADSR:
      jsr  readNext             ; read next song instruction/data
      ldy  $F6                  ; voice index
      sta  $D405,y              ; Generator 1: Attack/Decay
      jsr  readNext             ; read next song instruction/data
```

```
      sta  $D406,y               ; Generator 1: Sustain/Release
      jmp  finishInstr

;==================================
; set filter frequency
;==================================
setFilterFreq:                   ; set filter frequency
      jsr  readNext              ; read next song instruction/data
      sta  $D415                 ; Filter cut frequency: lo byte (bit 2-0)
      jsr  readNext              ; read next song instruction/data
      sta  $FF                   ; Transient data area of BASIC
      sta  $D416                 ; Filter cut frequency: hi byte
      jmp  finishInstr

;==================================
; set filter resonance
;==================================
setResonance:
      jsr  readNext              ; read next song instruction/data
      and  #$F0
      sta  $FC
      clc
      adc  $FA                   ; RS-232 output buffer pointer
      sta  $D417                 ; Filter resonance control/voice input control
      jmp  finishInstr

;==================================
; set volume and filter mode
;==================================
setVolume:
      jsr  readNext              ; read next song instruction/data
      and  #$0F
      sta  $FD
      clc
      adc  $FB                   ; Free 0 page for user program
      sta  $D418                 ; Select volume and filter mode
      jmp  finishInstr

     ; rts

;==================================
; set note lenght duration
;==================================
setDuration:
      jsr  readNext              ; read next song instruction/data
      ldy  $EB                   ; voice number
      sta  $00F2,y               ; note length duration
      jmp  finishInstr

;==================================
; change track position
;==================================
changePosition:
      ldx  $EA
      lda  $EC,x                 ; read position to go
      sta  $E0,x                 ; set new position
      lda  $ED,x
      sta  $E1,x
      jmp  finishInstr

;==================================
; set filter resonance (A)
;==================================
setResonanceA:
      ldy  $EB                   ; voice number
      lda  #$01
loopA:
      cpy  #$00
      beq  endA
      asl  a
      dey
      jmp  loopA
endA:
      ora  $FA
      sta  $FA
      lda  $FC
      clc
      adc  $FA
```

```
      sta  $D417                  ; Filter resonance control/voice input control
      jmp  finishInstr

;==================================
; set filter mode
;==================================
setFilterMode:
      jsr  readNext               ; read next song instruction/data
      and  #$F0
      sta  $FB
      clc
      adc  $FD
      sta  $D418                  ; Select volume and filter mode
      jmp  finishInstr

;==================================
; set filter resonance B
;==================================
setResonanceB:
      ldy  $EB                    ; voice number
      lda  #$FE
loopB:
      cpy  #$00
      sec
      beq  endB
      rol  a
      dey
      jmp  loopB
endB:
      and  $FA
      sta  $FA
      clc
      adc  $FC
      sta  $D417                  ; Filter resonance control/voice input control
      jmp  finishInstr

;==================================
; set number of repeat (A)
;==================================
setRepeatA:
      jsr  readNext               ; read next song instruction/data
      ldy  $EB                    ; voice number
      clc
      adc  #$01
      sta  $00D2,y                ; number of repeat factor
      jmp  finishInstr

     ; rts

;==================================
; goto the track to repeat (A)
;==================================
gotoRepeatA:
      jsr  readNext               ; read next song instruction/data
      tay
      jsr  readNext               ; read next song instruction/data
      ldx  $EB                    ; voice number
      dec  $D2,x                  ; dec repeat factor
      beq  skipRepeatA
      ldx  $EA
      sta  $E1,x                  ; change pointer
      sty  $E0,x                  ; to the next instruction
skipRepeatA:
      jmp  finishInstr

;==================================
; set various controls
;==================================
setControl11:                     ; set control voice to triangle
      lda  #$11
      bne  locSetControl

setControl10:                     ; set control voice to triangle (no A)
      lda  #$10
      bne  locSetControl

setControl41:                     ; set control voice to rectangle
      lda  #$41
```

```
      bne  locSetControl

setControl40:                   ; set control voice to rectangle (no A)
      lda  #$40

locSetControl:
      ldy  $F6                  ; voice index
      sta  $D404,y              ; Voice 1: Control registers
      jmp  finishInstr

;=================================
; set various duration
;=================================
setDuration2:                   ; set note length duration to 2
      lda  #$02
      bne  locSetDuration

setDuration6:                   ; set note length duration to 6
      lda  #$06

locSetDuration:
      ldy  $EB                  ; voice number
      sta  $00F2,y              ; set note length duration
      lda  #$40
      sta  $FE
      jmp  finishInstr

;=================================
; set number of repeat (B)
;=================================
setRepeatB:
      jsr  readNext             ; read next song instruction/data
      ldy  $EB                  ; voice number
      clc
      adc  #$01
      sta  $00D5,y              ; number of repeat factor
      jmp  finishInstr

;=================================
; goto the track to repeat (B)
;=================================
gotoRepeatB:
      jsr  readNext             ; read next song instruction/data
      tay
      jsr  readNext             ; read next song instruction/data
      ldx  $EB                  ; voice number
      dec  $D5,x                ; dec repeat factor
      beq  skipRepeatB
      ldx  $EA
      sta  $E1,x                ; change pointer
      sty  $E0,x                ; to the next instruction
skipRepeatB:
      jmp  finishInstr

;=================================
; read next song instruction/data
;=================================
readNext:
      ldx  $EA
      lda  ($E0,x)              ; read the value
      pha
      jsr  incForNext           ; increment pointer for next reading
      pla
      rts

;=================================
; adjust duration of note
;=================================
adjustDuration:
      lda  $F2                  ; note length duration voice 1
      cmp  $F3                  ; note length duration voice 2
      bcc  loc1
      lda  $F3                  ; note length duration voice 2
      cmp  $F4                  ; note length duration voice 3
      bcc  loc2
      lda  $F4                  ; note length duration voice 3
      jmp  loc2
loc1:
```

```
      lda  $F2                  ; note length duration voice 1
      cmp  $F4                  ; note length duration voice 3
      bcc  loc2
      lda  $F4                  ; note length duration voice 3
loc2:
      sta  $F5
      lda  $E1                  ; seg. of track 1
      beq  loc3                 ; jump if not active
      lda  $F2                  ; note length duration voice 1
      sec
      sbc  $F5
      sta  $F2                  ; note length duration voice 1
loc3:
      lda  $E3                  ; seg. of track 2
      beq  loc4                 ; jump if not active
      lda  $F3                  ; note length duration voice 2
      sec
      sbc  $F5
      sta  $F3                  ; note length duration voice 2
loc4:
      lda  $E5                  ; seg. of track 3
      beq  loc5                 ; jump if not active
      lda  $F4                  ; note length duration voice 3
      sec
      sbc  $F5
      sta  $F4                  ; note length duration voice 3
loc5: rts

instTable:
  .byte <setControl11
  .byte >setControl11
  .byte <setFrequency
  .byte >setFrequency
  .byte <setControl
  .byte >setControl
  .byte <setAmplitude
  .byte >setAmplitude
  .byte <setADSR
  .byte >setADSR
  .byte <setFilterFreq
  .byte >setFilterFreq
  .byte <setResonance
  .byte >setResonance
  .byte <setVolume
  .byte >setVolume
  .byte <setDuration
  .byte >setDuration
  .byte <changePosition
  .byte >changePosition
  .byte <setResonanceA
  .byte >setResonanceA
  .byte <setFilterMode
  .byte >setFilterMode
  .byte <setResonanceB
  .byte >setResonanceB
  .byte <setControl10
  .byte >setControl10
  .byte <gotoRepeatA
  .byte >gotoRepeatA
  .byte <setRepeatA
  .byte >setRepeatA
  .byte <setControl41
  .byte >setControl41
  .byte <setControl40
  .byte >setControl40
  .byte <setDuration2
  .byte >setDuration2
  .byte <setDuration6
  .byte >setDuration6
  .byte <setRepeatB
  .byte >setRepeatB
  .byte <gotoRepeatB
  .byte >gotoRepeatB

track1:
  .byte $38, $0F            ; set volume to 15
  .byte $20, $0F, $00       ; set ADSL
  .byte $00                 ; set control to 11 (triangle)
```

```
  .byte $08, $0F, $43           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control to 11 (triangle)
  .byte $08, $83, $59           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control to 11 (triangle)
  .byte $08, $C7, $70           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control to 11 (triangle)
  .byte $08, $0F, $43           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control to 11 (triangle)
  .byte $08, $45, $4B           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $B0

track2:
  .byte $20, $0F, $00           ; set ADSL
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $C1, $2C           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $D1, $12           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $A5, $1F           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $1E, $19           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $A2, $25           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $B0

track3:
  .byte $38, $0F                ; set volume to 15
  .byte $20, $0C, $00           ; set ADSL
  .byte $18, $00, $09           ; set amplitude
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $0F, $43           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $45, $4B           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $7D, $54           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $63, $38           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $3C, $32           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $98                     ; set note length duration to 6
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $C7, $70           ; set frequency
  .byte $40, $0C                ; set note length duration
  .byte $88                     ; set control to 40 (rectangle no A)
```

```
  .byte $98                     ; set note length duration to 6
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $79, $64           ; set frequency
  .byte $40, $12                ; set note length duration
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $B0

track9:
  .byte $38, $0F                ; set volume to 15
  .byte $20, $07, $00           ; set ADSR
  .byte $58, $10                ; set filter mode
  .byte $28, $00, $87           ; set filter frequency
  .byte $30, $28                ; set resonance
  .byte $78, $02                ; set number of repeat (A)
rBD60:
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $3C, $32           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $70, <rBD60, >rBD60     ; goto repeat (A)

  .byte $00                     ; set control 11 (triangle)
  .byte $40, $0E                ; set note length duration
  .byte $78, $05                ; set number of repeat (A)
rBD6F:
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $98                     ; set note length duration to 6
  .byte $70, <rBD6F, >rBD6F     ; goto repeat (A)

  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $40, $0E                ; set note length duration
  .byte $78, $02                ; set number of repeat (A)
rBD7D:
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $98                     ; set note length duration to 6
  .byte $70, <rBD7D, >rBD7D     ; goto repeat (A)

  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $78, $01                ; set number of repeat (A)
rBD88:
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $3E, $2A           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $C1, $2C           ; set frequency
  .byte $40, $0E                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $70, <rBD88, >rBD88     ; goto repeat (A)

  .byte $00                     ; set control 11 (triangle)
  .byte $08, $3E, $2A           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $C1, $2C           ; set frequency
  .byte $40, $2E                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $A2, $25           ; set frequency
  .byte $40, $16                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $78, $02                ; set number of repeat (A)
rBDB3:
```

```
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $3E, $2A           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $70, <rBDB3, >rBDB3     ; goto repeat (A)

  .byte $00                     ; set control 11 (triangle)
  .byte $40, $0E                ; set note length duration
  .byte $78, $05                ; set number of repeat (A)
rBDC2:
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $98                     ; set note length duration to 6
  .byte $70, <rBDC2, >rBDC2     ; goto repeat (A)

  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $40, $0E                ; set note length duration
  .byte $78, $02                ; set number of repeat (A)
rBDD0:
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $98                     ; set note length duration to 6
  .byte $70, <rBDD0, >rBDD0     ; goto repeat (A)

  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $78, $01                ; set number of repeat (A)
rBDDB:
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $A2, $25           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $3E, $2A           ; set frequency
  .byte $40, $0E                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $70, <rBDDB, >rBDDB     ; goto repeat (A)

  .byte $00                     ; set control 11 (triangle)
  .byte $08, $A2, $25           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $3C, $32           ; set frequency
  .byte $40, $48                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $48                     ; change position
  .byte $B0

track10:
  .byte $20, $37, $32           ; set ADSL
  .byte $50                     ; set resonance A
  .byte $18, $00, $08           ; set amplitude
  .byte $78, $01                ; set number of repeat (A)
rBE07:
  .byte $A0, $01                ; set number of repeat (B)
rBE09:
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $C3, $10           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $87, $21           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $A8, <rBE09, >rBE09     ; goto repeat (B)
```

```
  .byte $A0, $01                ; set number of repeat (B)
rBE1C:
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $D2, $0F           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $A5, $1F           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $A8, <rBE1C, >rBE1C     ; goto repeat (B)

  .byte $A0, $01                ; set number of repeat (B)
rBE2F:
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $EF, $0E           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $DF, $1D           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $A8, <rBE2F, >rBE2F     ; goto repeat (B)

  .byte $A0, $01                ; set number of repeat (B)
rBE42:
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $18, $0E           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $31, $1C           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $A8, <rBE42, >rBE42     ; goto repeat (B)

  .byte $A0, $03                ; set number of repeat (B)
rBE55:
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $4E, $0D           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $9C, $1A           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $A8, <rBE55, >rBE55     ; goto repeat (B)

  .byte $A0, $01                ; set number of repeat (B)
rBE68:
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $8F, $0C           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $1E, $19           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $A8, <rBE68, >rBE68     ; goto repeat (B)

  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $18, $0E           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $31, $1C           ; set frequency
  .byte $98                     ; set note length duration to 6
```

```
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $D2, $0F           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $A5, $1F           ; set frequency
  .byte $98                     ; set note length duration to 6
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $70, <rBE07, >rBE07     ; goto repeat (A)

  .byte $90                     ; set note length duration to 2
  .byte $48                     ; change position
  .byte $B0

track4:
  .byte $38, $0F                ; set volume to 15
  .byte $20, $00, $F0           ; set ADSL
  .byte $80                     ; set control to 41 (rectangle)
  .byte $18, $00, $08           ; set amplitude
  .byte $08, $F7, $09           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $8F, $0A           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $68, $09           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $8F, $0A           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $8F, $0C           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $4E, $0D           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $18, $0E           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $C3, $10           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $EF, $13           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $60, $16           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $31, $1C           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $DF, $1D           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $86, $23           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $16, $22           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $87, $21           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $02, $24           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $86, $23           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $3B, $25           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $02, $26           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $86, $23           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $08, $20, $24           ; set frequency
  .byte $40, $01                ; set note length duration
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $B0

track5:
  .byte $38, $0F                ; set volumne to 15
  .byte $20, $0F, $00           ; set ADSL
  .byte $80                     ; set control to 41 (rectangle)
  .byte $18, $00, $08           ; set amplitude
  .byte $78, $01                ; set number of repeat (A)
rBF1A:
  .byte $08, $8F, $0C           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $08, $8F, $0B           ; set frequency
```

```
  .byte $40, $01                ; set note length duration
  .byte $70, <rBF1A, >rBF1A     ; goto repeat (A)

  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $90                     ; set note length duration to 2
  .byte $80                     ; set control to 41 (rectangle)
  .byte $18, $00, $08
  .byte $08, $5F, $10           ; set frequency
  .byte $40, $08                ; set note length duration
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $B0

track6:
  .byte $38, $0F                ; set volume to 15
  .byte $20, $4F, $F4           ; set ADSL
  .byte $18, $00, $08           ; set amplitude
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $68, $03           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $89, $03           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $AE, $03           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $C8, $03           ; set frequency
  .byte $40, $04                ; set note length duration
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $AE, $03           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $80                     ; set control to 41 (rectangle)
  .byte $08, $93, $03           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $88                     ; set control to 40 (rectangle no A)
  .byte $B0

track7:
  .byte $38, $0F                ; set volume to 15
  .byte $20, $2C, $C2           ; set ADSL
  .byte $78, $02                ; set number of repeat (A)
rBF69:
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $D6, $5E           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $C7, $70           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $97, $7E           ; set frequency
  .byte $90                     ; set note length duration to 2
  .byte $70, <rBF69, >rBF69     ; goto repeat (A)

  .byte $68                     ; set control to 10 (triangle no A)
  .byte $B0

track8:
  .byte $38, $0F                ; set volume to 15
  .byte $20, $0C, $00           ; set ADSL
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $1E, $86           ; set frequency
  .byte $90                     ; set note length duration to 2
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $8B, $96           ; set frequency
  .byte $90                     ; set note length duration to 2
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $2B, $9F           ; set frequency
  .byte $90                     ; set note length duration to 2
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $FA, $A8           ; set frequency
  .byte $90                     ; set note length duration to 2
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $30, $AB           ; set frequency
  .byte $90                     ; set note length duration to 2
```

```
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $06, $B3           ; set frequency
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $FF, $5E           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $8B, $5A           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $20, $0C, $74           ; set ADSL
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $64, $56           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $68                     ; set control to 10 (triangle no A)
  .byte $B0

track12:
  .byte $10, $00                ; set control
  .byte $08, $00, $00           ; set frequency
  .byte $B0

track11:
  .byte $38, $0F                ; set volume to 15
  .byte $20, $07, $60           ; set ADSL
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $79, $64           ; set frequency
  .byte $90                     ; set note length duration to 2
  .byte $78, $01                ; set number of repeat (A)
rBFD0:
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $C7, $6F           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $C7, $69           ; set frequency
  .byte $90                     ; set note length duration to 2
  .byte $70, <rBFD0, >rBFD0     ; goto repeat (A)

  .byte $00                     ; set control 11 (triangle)
  .byte $08, $79, $64           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $78, $02                ; set number of repeat (A)
rBFE6:
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $0F, $43           ; set frequency
  .byte $90                     ; set note length duration to 2
  .byte $00                     ; set control 11 (triangle)
  .byte $08, $4B, $3F           ; set frequency
  .byte $40, $03                ; set note length duration
  .byte $70, <rBFE6, >rBFE6     ; goto repeat (A)

  .byte $68                     ; set control to 10 (triangle no A)
  .byte $48                     ; change position
  .byte $B0
```


Conclusion:

Ripping this game was a good experience for me. I hope that this description have let you achieve some idea of how to rip an old game: there are other games that attend your rip!!!

Finally I like to thanks Petri Keränen that for sure was happy for this rip

SSDin 1 end