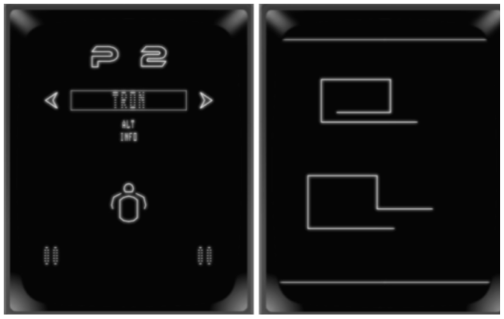
PLAYER 2

Vectrex-style packaging

The perfect 80's style package contains all sorts of treats for the perfect retrogamer and collector: an instruction booklet in 5 languages, game stickers, a nice poster and even the coloured transparent overlay to put on the screen of your beloved console. All quality stuff that will make you think you've just bought your favorite game in a little computer store of the Eighties.



Lots of games for 2 challengers

With 10 "classic" titles ready to play there is really no risk of getting bored. The selections menu is comfortable and fast. The cartridge content also features many extras, 4 hidden games, a bonus application, 4 screensavers, several easter eggs to be discovered and even a hardware test.

GAME RATING

PLAYABILITY

90%

The overall playability of the many games is Player 2's main point of strength. Being able to play two-player games in itself increases the fun and for each title the winning technique has been taken care of down to the last detail. A playful experience with a lot of atmosphere which reveals itself to be "magical" in a sense, a bit like the console the cartridge runs on.

LONGEVITY

95%

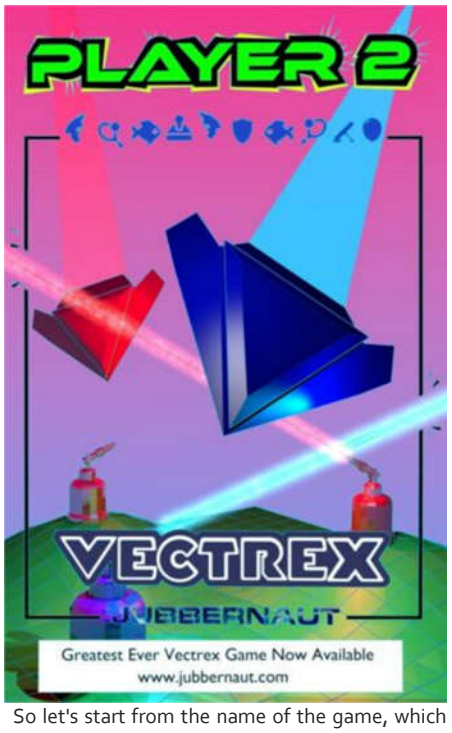
The high number of games packed throughout the cartridge ROM space gives Player 2 a special place in the software library of any Vectrex user. Beating your opponent is not enough. You also must try to do it with the highest score. As if to say: a challenge within the challenge, that will keep you glued to the screen!

Player 2

Jubbernaut – Year of release: 2018 – System: Vectrex

It was about a year and a half ago when Robin Jubber told me for the first time about his project to start programming some small games in 6809 assembly for the Vectrex, which incidentally is without a doubt the most "enigmatic" console in the history of video games, mainly for its particular design and charm that, even today, manage to attract many fans of retrogaming. At the time, I must confess, I thought that his plan was a bit crazy or at least definitely out of the ordinary. However Robin is an Englishman so that's nothing to be surprised about! Moreover, the virtue of eccentricity and lucid madness is undoubtedly one of the hallmarks of programmers, especially the creative ones who devote themselves to the design and the creation of video games. So, after all, there is nothing new under the sun in the world of hard-coding, the solemn and deserving practice (now virtually dying out) in which it is customary to squeeze a system to the last cycle. Robin is in fact a professional programmer of video games in his normal life, while during his free time he does something else: err... well, not exactly, because when he's not at work, where he usually remains buried under a ton of C/C++ code or basically merged with the Unity development platform, well, Robin... writes more code and games for his beloved 8/16 bit computers, such as the BBC Micro, the Acorn Archimedes and now the Vectrex! And even if I don't know him very well, just after his first e-mail messages I realised that he belongs to the Sacred Round Table of the Knights Programmers In Shining Armour who, when confronted with new obstacles and impossible challenges, throw themselves forward recklessly, ignoring all common sense rules. And I was not wrong. About six months after his first message that started with "Hey, you know I'm taking a look at 6809 assembly and how to write a game for Vectrex", he actually announced me that he was about to finish a new game for the Vectrex. As a matter of fact, there were 9 different games in his first work and he had even managed to store them all in one single cartridge!

You can read about his adventures discovering this mysterious console in this same issue of RM (see his own article titled "Never buy a Vectrex!"), while I'll talk here about his "multi-game" called Player 2, released last October by Jubbernaut, the software house that Robin himself set up to distribute the products of his labors in an efficient and old-fashioned way, with packaging, instruction booklet and accessories included. In fact, if you ask him, he will tell you that the most complicated part of the game production was putting together the packaging and the shipping or finding the cartridge plastics and the coloured transparent overlays to be glued to the screen, certainly not the coding sessions! And he started essentially from scratch with 6809 assembly and the console OS! What did I tell you? Basically a fool! ☺ But the passion for programming and game design, combined with a nostalgic desire to write code and publish artfully packaged games, this made "the miracle" come true. Robin really did manage to produce a video game that is extraordinary for the Vectrex and certainly unique.



So let's start from the name of the game, which obviously was not selected by chance. The cartridge contains, as mentioned, as many as 10 complete games, all inspired by the console/arcade classics of the early 80s. The titles included in Player 2 must be played strictly by 2 players, as in an endless challenge for the (unfortunately) rare owners of two gamepads. The list is very rich: ranging from Pong/Tennis to Tron, from a clone of Asteroids (Rotatazor) to one of Pit Stop (Racers). There are also battleships between the stars (Space War) or classic shooting games (SpiderFish, Balloons and Artillery). The more extensive game called Incoming is a real novelty and ends with a bang the list of available games. In this game two players are called to collaborate with each other, going through a puzzle adventure of 100 screens and 7 different levels full of pitfalls and dangers, doors and traps, aliens ready to attack, magnets, switches, fans and many other obstacles. Our goal is to lead a critical energy cell to the end of the maze of rooms and locations. Incoming is the only game that can be played by a single player as a time trial version is included. Until now, few Vectrex titles allowed 2 players to compete on the screen at the same time. With the appearance of Player 2 this gap has definitely been filled. Some games are reasonably easy to play, others are pretty difficult and others frankly impossible. At least for me and my daughter, during my first days of attempts. But the fun is guaranteed: in its apparent simplicity, the console hardware is properly exploited by Robin's code, who used, as he discovered them while proceeding, advanced techniques to overcome many graphic limitations, slow computing power, small space available on the cartridge and even absurd bugs of the basic operating system.

For more information and to purchase the game please refer to: http://www.jubbernaut.com

by David La Monaca (Cercamon)

Don't ever buy a Vectrex!

by Robin Jubber (layout and adapting by David La Monaca/Cercamon)

Don't buy a Vectrex! Sure, without one your console collection is just a bunch of plastic landfill, and yes, it's the greatest console of all time; but still. Don't ever buy a Vectrex! I bought one and now I have three. This vital information has been withheld from the girlfriend, and because her knowledge about retrocomputing is basically null, she may never find out, god willing. I also ended up with a stack of expensive homebrew, a ton of wonky peripherals and I recently wrote a game – well, 14 games – for the machine and now my house is full of packaging, soldering irons and plastic carts. I've become a home based manufacturing company – I should have bought a Neo Geo like a sane person.

If you're unfamiliar with the Vectrex, don't feel bad. Certainly it means you're not a proper console collector, know nothing about retro gaming and shouldn't be reading this magazine but on the other hand, your bank balance probably isn't a giant smoking hole in the ground. The Vectrex is a rare and unusual beast. It's the only non-portable with an integrated screen. It's the only console that uses vectors instead of pixels. It's one of a few select consoles where a second controller is almost as expensive as the console itself (that's why I bought the third machine, honey). It has colour graphics, but only in the weirdest possible sense. It was the first console with a 3D headset. Every one comes with slightly wonky graphics, but in a manner unique to that machine. It makes a weird buzzing noise even with the volume at zero. Opening it up for repairs can literally kill you. It has a carrying handle of sorts, presumably so you could take it very, very carefully to your friend's house but the carrying handle is incomplete and sort of sloped, so dropping it is a strong and expensive possibility. It was only in production for about a year and a half. It has around 800 bytes of ram. It is, without doubt, the greatest console of all time.

The first thing you'll notice is the screen. Essentially an old black and white CRT on its side, the vector display is like nothing else out there. The simple line-based graphics pop out from the screen in a way that cannot be replicated by an emulator. Your 8 gig graphics card cannot make graphics like this. Older readers may have played the original vector Star Wars in the arcades – this is essentially the same technology, squeezed into something that sits on your desk. Because the graphics are so unusual, and don't use pixels at all, my second mistake was to try writing a little code. Perhaps I could get a simple triangle up on screen. That could be fun. Then I'd get back to writing proper games on machines with millions of kilobytes of ram instead of not quite 1. A little searching on the internet turned up a few snippets of information, such as Christopher Tumber's 1998 text file that helps with some aspects

Figure 1 – The Vectrex in action!

of the machine and 6809 assembler. You'll also need as09, for turning code into binary, and ParaJVE, a not entirely accurate or finished Vectrex emulator. It's also possible to write your code to eeprom and test it that way, but you'll die of old age before you get Hello World world up and running. There is also, incredibly, a full development environment created by a lovely chap called Malban, but I didn't want to bother with that – this was just meant to be a quick experiment after all. But we've already established I may be an idiot.

The 6809 that sits at the heart of the machine is a wonderful cpu. It really only turns up in embedded systems, the Vectrex, the Dragon 32 and the TRS-80. And that's a shame. The architecture is big endian, which means a lot of 16-bit maths is easier for humans to understand. Little endian, which turns up in rival 8-bit cpus, is more convenient for simple logic units to handle a byte at a time, which keeps manufacturing costs down, but doesn't make for such readable or intuitive code. For a supposedly 8-bit cpu, the 6809 is also very 16-bit capable. It has two 8-bit registers that combine to create a 16-bit word, along with a host of other 16-bit registers and instructions. Most of the code you write will be 8-bit, the usual business of comparing small values and looping over small data structures – but when you need to write some fixed point maths, the 6809 really comes into its own. For my first experiment, a simple Pong clone, this wasn't all that important. By the third game, a simple Spacewar clone, it had become vital to making smooth movement. Essentially you don't want to be restricted to integer mathematics – you can then only move discrete numbers of units per frame – and as the Vectrex only understands 256 units vertically and horizontally, that would mean your minimum movement rate would cross the screen in just a few seconds. So you use 16-bit values, and treat the first byte as the actual coordinate, and the second byte as the fractional part. Very easy with a 6809.

The problem with writing a simple Pong clone is that I still had about 25K of rom space free. Obviously I could have just released the binary file and got on with the day job, but by now I was hooked on understanding what the Vectrex could do and learning a new language. Plus Joanna, my baby daughter, had just turned up out of nowhere so I was essentially on self-imposed paternity leave. So I wrote another game. And another. Then I shuffled memory around a bit, noticed some routines were being used all over the place and found some more spare rom space. So I wrote another game. At this point the original 32K rom, which is all the Vectrex can address, was getting close to the limit, so my four or five simple games would end up on the internet and I could move on, having scratched a programming itch. I'd also mentioned my Vectrex adventures to some other ancient programmers who've been making games since the dark ages and they were very jealous. No compile times. No producer leaning over my shoulder. No gently explaining to simple minded artists why their artwork is all broken and useless. Just raw assembler,

written directly to the metal – proper game coding like we used to do when men still lived in caves.

At this point things took a turn for the worse. I discovered that some clever coder had figured out how to do bank switching on the Vectrex. You could have 64K of code and data, provided you were very careful about how it was initially laid out. If the first chunk of instructions are the same for both banks, you could send a signal to a magical place in the Vectrex hardware, and switch to bank 2. Or back again. Using the same technique, you could also write to a 32

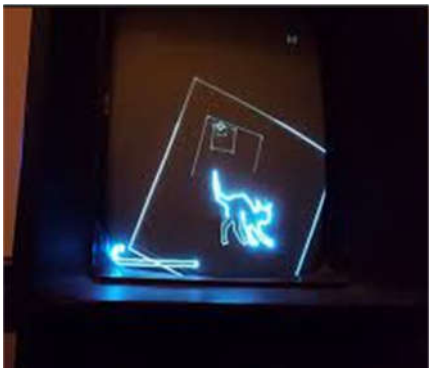


Figure 2 – Drawing with the Vectrex

character eprom, in effect giving the Vectrex save and load capabilities. Well, obviously I had to have some of that. Another 32K to play with? This could become a compendium of two player games. Two player games have a couple of distinct advantages to the lazy incompetent coder. Firstly, the Vectrex simply doesn't have much in the way of two player gaming options, despite two joypad ports. Many games can be played with alternating players, but if you want to show off your fancy new Vectrex to a baffled friend, you need proper simultaneous two player. The other advantage is that I wouldn't need to write much in the way of artificial intelligence. And to think, my computer science teacher worried I didn't have the right mindset to be a programmer. What a fool! Sure, if I had done things his way I'd be writing database software for a bank and driving around in a Lamborghini, instead of a 13 year old BMW that likes to shut down in the middle of the road for no apparent reason while everybody points and laughs. But I wouldn't get to make spaceships move around on screen, so it's a fair compromise. I think.

By now I was up to 6 or 7 games, including my first properly complicated bit of code, for a single screen worms-style artillery game. Next came a basic stock car game, played from overhead and then Tron. The Vectrex is all about lines so Tron seemed a natural fit. In theory. The unusual thing about the Vectrex is that it has no persistence. Nothing you draw stays on screen for more than a frame – everything has to be redrawn 50 times a second, completely. Nothing else works like this, although modern 3D games essentially have to address the same issue. You have 30000 cpu cycles to get everything drawn and then the entire process has to start again as the previous frame's lines fade quickly from view. The Vectrex draws graphics in the style of an oscilloscope – the electron beam leaves a trail but it doesn't persist for long. The scaling of the lines you draw also introduces delays and text is especially hard. The OS routines for displaying text are not fast, and grow increasingly distorted the more letters you display. Your 30,000 ticks are constantly under pressure. With Tron I couldn't write the game in the usual manner, by adding a pixel to the head of the snake, and simply deleting a pixel from the tail. Instead I had to store all the lines, including the special case of lines that are longer than 128 units (the maximum on the Vectrex) and also keep extending the line forwards as the player moves around the screen. This set of lines then had to be packaged up to the OS as a complete vector image, so it could be drawn instantly, for both players. Well, no problem – I have a few hundred bytes free, if I wipe over ram that is used by other games but which Tron doesn't need. That meant the missile system, the explosion system, the block based collision system, the particle system – they can all be repurposed. I just need a collision system that handles line intercepts and a giant queue to handle the line ends. Also two of everything because there are two players. Oh my god that is some ugly code – but it works. But still – all that rom code left to fill. And I had better fill it fast or my daughter will have left for university by the time this cart is done. So I took the Spacewar code, added gravity, and built a series of caverns for the two ships to navigate. A co-op game on the Vectrex! With 100 rooms! Those 100 rooms nearly killed me – level design is soooo tedious, but eventually it was up and running. I won't bore you with the critical bugs, the realisation that saving and loading also force a bank switch, the easter eggs I added which nearly broke everything and the amount of data shuffling needed to fit 5 translated languages into the cart. I also had Malban look over the code - "it's very good for a first game, but you probably shouldn't have written something this large and here are all the things wrong with it" - where he went through 22 thousand lines of assembler looking for places where I incremented a loop instead of decrementing it (saving, 1 or 2 bytes) or loaded two registers separately instead of one (saving 1 byte). The man's a mad genius and I'm not ignoring help like that – it all adds up. Bytes and clock cycles are precious when you're coding in the past. Malban also helped me understand the relationship between the scale of vectors

Figure 3 - The Vectrex in all its shining!

and their drawtime, which was invaluable for getting the menu system to run without dropping frames.

So the game is done, except for the hidden Black Vector screensaver which I wrote at the last minute, and the hidden message to my girlfriend (because every girl dreams of getting a feeble apology delivered on antique hardware) and the other hidden features I add because I can see some free bytes just lying around doing nothing. Now I'm up to 14 games, 6 screensavers, 10 utilities, 6 game settings, 5 languages, 11 alternative versions for the main games, 3 bits of hidden seasonal DLC and I get that twitchy feeling when the project is finished and you want to write some more code but there's nothing to write and nowhere to put it anyway!

And that's just the start of my problems. Right now I'm a manufacturing hub because to make a Vectrex game properly you need to create carts, burn roms, solder components onto pcbs, you need to find somewhere that can make boxes, you need to make some artwork, write manuals and their translations (many thanks to all who helped with that), draw posters for that proper 80s feel, edit a web page and that's before you even get to packaging everything up and talking to all your customers. I also had to find a friendly plastics manufacturer and carefully explain what a Vectrex is, and why I need a colourful two layer piece of polycarbonate with a pretty pattern on it. My first step towards all this was buying a 3D printer in order to make carts. Test cart 1 looked like that bit in Judgement Day where the T-1000 is trying to reform after Linda Hamilton blasts him with a shotgun. My next attempt resembled the special effects from the sci-fi classic The Thing. I'm a coder – I don't know anything about plastics! Now I buy my carts from America where it merely costs all the money. So hopefully this will give you a slight idea why you shouldn't buy a Vectrex. I'm serious. You have no idea what you're getting yourself into. Even if it is, without doubt, the very greatest console ever made.


```
; Define some OS routines we will need
OS Intensity 7F  equ      $F2A9    ; sets the intensity of drawing to maximum
OS Intensity A   equ      $F2AB    ; this version uses the a register
OS Wait Recal            equ      $F192    ; Vectrex BIOS recalibration
OS_Print_Str_D   equ      $F37A    ; BIOS print routine
OS_Reset0ref             equ      $F354    ; Call frequently to avoid wobble
OS_Move          equ      $F2FC    ; move to location specified by registers a and b
OS_Draw          equ      $F3DF    ; draw by amount in registers a and b

; some assembler directives (optimise and start of code in memory)
        opt
        org      0

; HEADER SECTION
        db       "g GCE 2018", $80
        dw       $FF8F                    ; address of a (BIOS ROM) tune
        db       $FC, $30, $20, -$58      ; height (negative), width, rel y, rel x
        db       "JUBBERNAUT", $80        ; title: db sets aside some bytes in ram
        db       0                        ; end of game header

; GAME
Main
        jsr      OS_Wait_Recal    ; start of draw cycle
        jsr      OS_Intensity_7F  ; set the intensity of the beam - 7F is max

Draw_A_Line
        jsr      OS_Reset0ref     ; Reset the beam
        lda      #30
        ldb      #-64
        jsr      OS Move          ; Move to -64,30 (a and b are y and x)
        lda      #0
        ldb      #125
        jsr      OS_Draw          ; draw a horizontal line (0 units in y, 125 in x)

Write_Some_Text
        jsr      OS_Reset0ref
        lda      #0               ; OS_Draw corrupts registers a and b
        ldb      #-60
        ldu      #MESSAGE         ; load the 16-bit U register with the string
        jsr      OS_Print_Str_D   ; call the OS print routine

Draw Another Line
        jsr      OS Reset0ref
        lda      #-30
        ldb      #-64
        jsr      OS_Move
        lda      #0
        ldb      #127
        jsr      OS_Draw
Main_End
        bra      Main             ; and return to the start of the game

MESSAGE  db       "ROBIN IS SKILL!",$80

; the Vectrex font is capitals only. Some punctuation and special symbols
; are also supported.
```

Listing 1 – "Hello World" coded in assembly for Vectrex/Motorola 6809

Vectrex Technical Addendum

Here are the minimal steps to build your own environment and write your "Hello World".

1. Download the compiler as09.exe from this LINK or try VIDE by Malban, which has an assembler built in.

2. Choose an IDE. I initially went with Visual Studio and created a rule in **Tools -> External Tools**. Add a new external tool and call it Vectrex Assembler. For **Command** enter **C:\Wherever\as09.exe and for Arguments use –q $(ItemFilename)**. Tick **Use Output Window** and leave the other options blank. You will also want to add a keyboard shortcut, usually found in **Options**, to execute this external tool on the current file. I use **F6**, but your tastes may vary. Of course you can skip all this and just use VIDE, which has buttons to compile assembler into 6809.

3. Run your code. As09 will output a bin (binary executable) file which you can load with ParaJVE, a Vectrex emulator. ParaJVE can also handle bank switching, but not save/load – although these are features most Vectrex games do not use or need. Again, VIDE has an emulator inside the IDE, and it's more accurate for spotting framerate and draw problems.

4. Write some code. Each Vectrex game has a header, which will be displayed by the OS on bootup. After that header is where your code begins. An example is provided in listing 1.

5. Become addicted to 6809 assembler and watch your social life disintegrate.

Some handy hints for Vectrex development I wish I knew beforehand:

- The OS routines for printing strings will fail with no explanation if your string is just one character long. You won't have a clue why.
- The longer a string, the more distorted it will become. In many cases it is better to scroll a string than display it as one page of text.
- When designing graphics, for instance in Blender, try and make your vector sprite as large as possible – generally to fill the screen – then scale it down to display it. The smaller the scale of a vector list, the less cycles it will take to display. There are subtle aspects to this guidance, but in essence this is a good rule of thumb.
- ParaJVE draws as if running on a magically perfect Vectrex. No such beast exists in the wild.
- Any attempt to save or load will cause a bank switch. Have all your EEPROM, initialisation and bank switching code in identical locations in their respective banks.
- Don't worry too much about writing super efficient assembler code, except inside critical loops. Vector redraw is going to be your framerate killer, unless you're doing a lot of 3D maths. In modern parlance, you're most likely to be "GPU bound".
- On the 6809, most instructions, for instance loads, will also affect the status flags.
- This page has a summary of the OS commands and the registers destroyed after jumping into these subroutines [LINK] - which is why a and b will no longer hold their values after moving the beam.
- When using the joysticks, you have to turn their respective features 'on' – for instance movement in x and y, and analogue control. If you don't need these facilities, leave them off and you will save a bit of cpu time.
- Before displaying text at the start of the frame, perform a move, otherwise the text will not show up.
- Going back to 6502 or Z80 will *hurt* after coding this chip.