# The Making of Bruce Lee II

*This article was originally published in Vandalism News #64 and has been reprinted with kind permission from Jazzcat. The original article can be found here: http://csdb.dk/release/?id=138976*

## Background

My parents bought a C64 as a Christmas gift to the whole family in 1983 but us kids were not allowed to play on it, only do serious stuff. For that reason it took a while to get to the point where we had a lot of games for it. I spent most of the time learning how to program instead. In my Mickey Mouse diary from 1984 I wrote every day about games I was creating. Eventually, we got access to games and I played Bruce Lee some time in the mid 1980s. I loved the game. It was an adventure to play. It was mysterious and I felt like anything could happen. I played it to the end, like I did with a lot of games back then. I actually had a list of completed games on one of my diskettes since achievements and trophies weren't invented yet.

I first read about Bruce Lee II in July 2013 in a thread on the Lemon64 forum. It was a poll asking if people wanted Bruce Lee II to be ported to a real Commodore 64. Someone had done a spiritual successor to the original Bruce Lee game but for PC. I tried it and saw that it had a mode where you could make it look like it was run on a C64. People in the thread were trying to organize development of the C64 conversion when I dropped in.

At that time I was working on the Cosmos conversion with my brother and didn't have time for anything else at all. But in time Cosmos was finished and we started talking about what to do next. We were talking about Scramble or variants of it with procedurally generated levels. I was thinking about a point-and-click adventure. After a while we agreed on Bruce Lee II since the attempts to start a conversion in the Lemon64 thread were cooling down and it looked like a fairly simple conversion with its flick screen scrolling. Cosmos was so much about optimizing screen clearing and software sprite drawing that it felt easy to do static backgrounds.

## The Prototype

I prepared a new empty development project and started to move reusable code from Cosmos over to a core library where the games could share code. I worked in Sublime Text and used DAsm to assemble the code. DAsm is the only assembler I found that can have macros with arguments that aren’t defined in the first assembly pass, so I’m sticking to that for now. I did all quick iteration testing in Vice and sometimes sent the stuff over to my C64 using a custom MIDI server solution to try it on my bread box machine. I use Mercurial for source control and I have a Mac Mini as the main server to share the code with my brother. I have an automatic backup using Pulse to a third computer and do manual backups to a cloud service to be pretty sure nothing is lost.

After a while my brother told me that he wouldn't be able to put enough time into the project, so he wanted to back out. I decided to continue on my own since I had stated that we were taking over the conversion in the Lemon64 thread and I didn't want to disappoint anyone.

I started out with C64 graphics that Mase from the forum had done and I used it to do my first iterations of screen drawing. After a while I realized that to be able to modify the data format it would be better to convert the

data from a more generic format. So I made PNG images of the original graphics which e5frog on the forum received from Bruno R Marcos, who made the PC version, and created a conversion script that baked the format I wanted. I knew that bitmap was necessary for at least some of the screens so I used character mode wherever possible and bitmap where needed, since bitmap is slower to draw. Some screens weren't even possible to reproduce in bitmap mode. The water in the PC version was made using a semi transparent blue rectangle on top of standard C64 colors so that was just not possible to get exactly right.

## Graphics Pipeline

I created a level editor using C# and WPF for the physics information, doors and items you can pick up. This again made it easy to change the output format later in the project and it was changed a number of times. I made a conversion script to convert the level editor file into something that the game could use effectively. I made a sprite conversion script as well. Most of the Cosmos sprites were made in SpritePad but the Bruce Lee II characters would not fit in one sprite image each so it would be difficult to work like that. Instead I converted PNG files exported from Aseprite where I recreated the animations. The conversion step cut sprites from the PNG files to create lots of sprite images. After some compression tests done on the screen data I realized this wouldn't fit in memory so I started planning streaming the graphics while the game is playing.

## Graphics Streaming

To make streaming work I needed a turbo loader, so I wrote one based on the Covert Bitops 2-bit loader. I wanted maximum compatibility so I didn't use the ATN bit in the loader because you have to turn off all other drives while playing the game if that is used. The downside is that sprites mess up the timing if they are drawn by the VIC-II at the same time as the loader reads data from the drive. I solved this by only loading while the VIC-II was drawing the border and only if there were no sprites in the border. Occasionally, Bruce can be partly outside the screen but luckily not for too long so that turned out to work really well.

I created a streaming system with trigger boxes that indicate the rooms that needed to be in memory. When the trigger boxes are entered, the room data is streamed in using the turbo loader. To avoid all waiting time, four compressed rooms are allowed to be in memory at the same time. There can also be conditions on the trigger boxes to reroute the streaming when the drawbridge goes down early in the game.

I sketch ideas on paper or in the notes app in my 3DS when I’m programming. Illustration 1 below shows a scan of an early streaming editor sketch. The blue lines divide the screens and you can see the red trigger box and the three adjacent screens to keep in memory when the trigger box is entered.

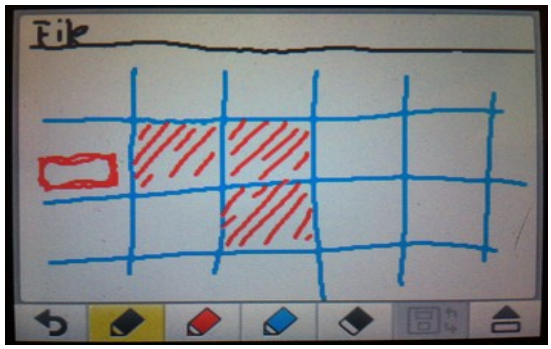


*Illustration 1: Streaming editor sketch*

To get everything properly converted I recorded myself playing the PC game so I could go back and measure pixel movement and durations to get everything right. We did that when converting Cosmos and it was really boring at first but very handy in the end. The only problem was that the PC

game had a bug that made it impossible for me to finish the game. Bruce gets stuck when entering the room the second time where he gets caught by Tao-Bao. I tried it on both my PCs with the same result. Finally, I used a Youtube playthrough for the last rooms so if there is anything wrong there, you know why.

## Threading

Just like Cosmos, memory became the biggest challenge. Supporting both NTSC and PAL wasn't a memory problem to begin with but memory restrictions heavily influenced the design decisions.

Cosmos was my first attempt to make an NTSC compatible game, or rather PAL compatible since the version me and my brother ported was running in 60Hz. The Cosmos game update loop was running in sync with the TV frequency so to make everything run at the correct speed, all moving objects needed to have specific speed values for NTSC and PAL. This was cumbersome. There were no simple pixel movements anywhere. Everything had to be done with subpixel precision. Objects moving in the same direction at the same speed also needed to be synchronized (having the same fractional position), otherwise there was a rubber band feeling to the movement. Also sound effects needed to be tweaked to get the same sound (frequency and timing). The whole thing was solved in Cosmos by having all NTSC specific data in a big continuous block and using it directly. If a PAL system was detected the whole block was overwritten with the PAL specific data. That way, the code didn't need to know the system, nor do branching on the system type and could read the settings from a fixed address.

This way of handling the different platforms was the first thing I wanted to change after Cosmos so when I started to work on Bruce Lee II I planned this from day one. I wanted the game logic to run at the same speed on both systems. That would make it trivial to get the same movement speed, music and sound effects on both NTSC and PAL. A timer interrupt that triggers the next game update was set up to run at 50Hz. The update checks for input, moves characters, checks collisions and triggers rendering. This was all fine if it wasn't for the single buffering. This is where the memory restrictions come in.

The bitmap mode consumes so much memory that it can't be double buffered. So, with a single frame buffer, some kind of technique must be used to synchronize screen changes with the TV frequency to avoid tearing. To solve this I stored graphic update information in a circular command queue. This is basically my double buffer. Putting stuff in the queue can be done at any time by the main loop. Fetching the data, updating sprite registers and drawing on the screen are done in sync with the screen refresh by a raster interrupt. When the raster has reached the bottom border, commands are fetched from the queue until the raster reaches the top border. Any outstanding jobs will need to wait for the next frame, but that rarely happens. Then it is time to handle the status bar split where hires and multicolor are switched and sprites are hidden behind the bar. This is similar to how modern graphics cards use command queues to store render states, with the difference that they will be read and written in parallel on modern hardware. This still has the potential for creating threading issues on the C64 with the raster interrupt interrupting the main update at any time, but it is fairly easy to handle this if the raster interrupt is only allowed to pull from the queue and write to screen or sprite registers while the main update is only allowed to push to the queue and not update the graphics. In general I am separating rendering from game logic to be able to turn off rendering for a frame in case the game

update takes too long and needs to keep up.

Illustration 2 shows an early sketch of the thread handling on NTSC. The upper line represents the render thread, triggered by the raster interrupt. It executes the jobs and updates the turbo loader. The lower line represents the main thread which is triggered by the timer interrupt and updates the game state.

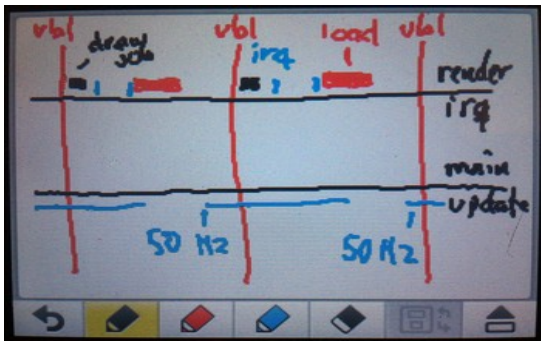


*Illustration 2: NTSC thread diagram*

## Animation

There are 138 frames of character animations in the game and some more frames for moving objects. Most characters require several sprites on top of each other to replicate all colors and pixels so in total there are around 350 sprite images in the game, which would consume over 20kB of memory. To fit all that in memory some kind of compression was needed. The first optimization was to only store character images for one direction and generate the opposite image in code. Bruce's sprite images are generated in real-time while he's moving around by using double buffered sprites. The sprite conversion script has the useful property that all non-empty sprite information is stored at the top left corner of each sprite image. I used that to cut off the empty space below all sprite images and stored the useful information along with the height of the sprite images. I then regenerated the empty space when decompressing in code. That wasn't enough either so I interlaced the sprites horizontally in memory. A sprite image is three bytes wide and if the rightmost byte is never used for any line in the sprite for example, I can basically put another sprite in that space if it is only using the leftmost byte for all lines. Those simple compression techniques together reduced the amount of memory needed to store all sprite images in memory a lot. Now, they also need to be decompressed into memory readable from the graphics chip. All graphics must fit within 16kB of memory and that is cut in half by the bitmap graphics. Around 100 images can exist at the same time in graphics memory and that isn't enough for four running characters at the same time. I handle this by keeping the standing and falling postures for all enemies in graphics memory at all times, so they can turn fast and fall at any time. I keep the running animation in only one direction in graphics memory at a time so if an enemy decides to turn and run in the opposite direction, the running animation frames in the other direction needs to be decompressed first. The PC game has delays to make the enemy turn slowly so this is more than enough to cover the decompression time.

## Sound and Music

I made a custom music player for the Cosmos music. I reused this but modified it to support several songs and also sound effects that can allocate channels dynamically. I don't have a music editor yet so I make the music in the assembler source using simple macros, like play_note, wait or arpeggio. I didn't rip the music from Bruce Lee I because it gets more tricky to fill the memory completely when some things must exist at a fixed address. Instead, I recreated the song in my own player. To get it right I used a tool I made to reverse engineer sound creation. It allows me to load a SID file and record the SID registers to a REU memory expansion cartridge in the background while I'm analyzing what

happens each frame. Since everything is recorded to REU memory I can fast forward or rewind the song and step one update at a time to see exactly what happens. The music player is versatile enough to replicate everything. I did the same for the sound effects that existed in Bruce Lee I.

Some new sound effects were introduced in the original Bruce Lee II which weren't created with a SID from the start. It was a challenge to replicate those. I used the spectrum analyzer in Audacity to figure out the frequencies used and then I tried my best to make it sound similar. The hardest one was Tao-Bao's laughter. I initially planned to play it as a sample but since memory was tight I scrapped that idea halfway into the project.

The intro music was made by constructing sounds in M64 that I played with using my EPS-16+ keyboard and Sentech MIDI interface. I also did some composing using my guitar, which I suck at playing. I then typed the instrument data and notes directly into the assembler source using my macros. I took two days of vacation from work to do this and it took me four days to complete. I made several attempts which I threw away before the final piece took form.

## Code Streaming

I worked on gameplay features one screen at a time, from the first screen to the last, in the order they were played. The code grew and grew and after 20 screens or so I realized that memory would not allow all code to be in memory at all times. I had to stream that from diskette as well. I solved this by having two buffers of streamed code memory. One for the current screen or area and one for the next. I couldn't afford more buffers so it isn't possible to go "back" when streaming code. At some points in the game it is impossible to go back. That became the boundaries for the streaming code zones. One zone is kept in memory to be run and the next is streaming in. When the zone is switched, the newly streamed code buffer is used and the old becomes the next to stream in. This made it possible to cram the last features into the game and actually finish it.

## Diskette Layout

The level editor exporter script outputs the streaming files in the order they are loaded when the game is played. To reduce the number of track changes while the game is played (because of the sound), the files needs to be placed close to each other and also in the loading order. I created a python script to bake disk images from files with specific requirements to make it possible for all files to be stored continuously on the diskette. You can hear when you get close to the end of the game how the drive seeks back to the first files to load them just before the final battle to get ready for the next round.

## Closing up

Near the end of the project I was very tired. It takes considerable effort to put the last pieces together to form a finished and polished game and in the end I devoted myself entirely to getting Bruce Lee II finished. I tested the game extensively on all my hardware and other hardware through emulation. I considered building an automatic testing system that played through the game without me so I could concentrate on the last pieces without worries about breaking stuff. This system would be running at night chasing unexpected bugs. I never built that system, but maybe it will be something for future projects.

## Learning

During the course of this game I discovered a couple of quirks of the C64 that I didn't know before. They were not discovered by me first, I just didn't know about them

before. The first quirk was sprite ghosting. When Bruce was jumping or climbing up in the border the turbo loader reported read errors on PAL machines. It was strange because the sprite images weren't overlapping with raster lines where the turbo loader was running. The loader was running in the lower border and the sprites were displayed in the top border. Since there had to be something stealing CPU cycles from the loader, I made a test where I created a pattern of background color changes on screen while it was blanked. When I placed a sprite in the top border I could see a disturbance in the otherwise stable pattern at the top (even when blanked), but also at the bottom! After asking around I got the answer. The sprite y-position is 8 bits and so is the raster compare, even if the raster line is 9 bits. So a sprite with a y-position such that there is a line >255 whose lower 8 bits matches the y-position will create a ghosted sprite.

I learned about the 6581 SID bug causing the ADSR to lock up for 30 ms if switching from long release to short attack for example. I noticed this when Bruce was climbing. The climbing sound changed slightly just after Bruce came up from the water because the water splash sound effect had a long release. I handled this by adding a command in the music player that can be used to manually reset the voice a couple of frames before a note with short attack.

Perhaps the greatest difference from the PC version was the status bar. The reason it isn't inverted is because, in order to make it safe for Bruce to pass below it the lowest line must be drawn using a foreground color. I need some time to move Bruce's sprites from off screen to on screen during one line at the same time as the hires/multicolor switch and background color switch. It gets a bit messy if you allow sprites near it and also are trying to support accelerator cards and not rely on exact timing. So the black color in the status bar is the foreground color, filled with 1s. I thought that if black also was used as the color for multicolor pixels filled with 11s the color would still be black and I have the whole line to make the switch. It turned out that there is a quirk in the VIC-II chip where it will choose another color for one cycle if you switch from hires to multicolor. The color for the multicolor pixel 10 will be chosen instead of the expected 11 for one cycle. It turned up on the two leftmost characters in the status bar and I managed to hide it by setting both the upper and lower nibble of character memory used for bitmap colors to black.

After the game release I heard that this cycle glitch was still visible on some machines further to the right in the status bar. I can't reproduce this on my machines or in Vice so there must be some odd timing difference that I'm not aware of. This and other small things can certainly be improved, but it is time to move on. There would never be a next game if I just improved the existing one to perfection.

I'll see you later!

*Ps. I have been asked several times why I didn’t sell the game instead of giving it away. Well, it wouldn’t be right to make money on someone else’s name and someone else’s game. I’ll consider selling if I can create something original, but it is a nice gesture to give away your games. I’m happy if people enjoy them. I’m not in it for the money.*

# References


Aseprite - http://www.aseprite.org/
Audacity - http://sourceforge.net/projects/audacity/
Bruce Lee I - http://www.c64.com/games/90
Bruce Lee II - http://kollektivet.nu/brucelee2
Bruce Lee II PC - http://www.bruneras.com/games_bruce2.php
Cosmos - http://kollektivet.nu/cosmos
Covert Bitops Loader - http://cadaver.homeftp.net/rants/irqload.htm
DAsm - http://dasm-dillon.sourceforge.net
M64 - http://kollektivet.nu/m64
Mercurial - http://mercurial.selenic.com
Pulse - https://ind.ie/labs
SID Hard Restart - http://www.df.lth.se/~triad/krad/sidmidi.html
SpritePad - http://www.coder.myby.co.uk/spritepad.htm
Sublime Text - http://www.sublimetext.com
The Lemon64 thread that started it - http://www.lemon64.com/forum/viewtopic.php?t=48721