ReGame 64

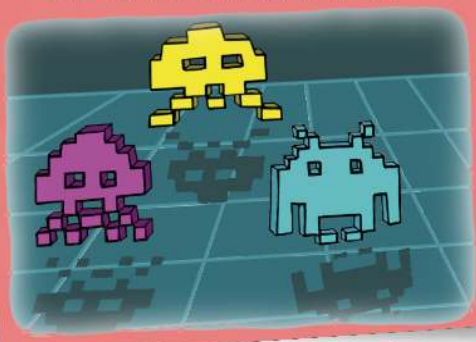
IT IS NEVER TOO LATE FOR A COMMODORE 64

The point-and-click adventure you ever dreams!

Limbo (?)

Donkey Kong 2016

Let's Invade!



IVAN VENTURI'S BOOK

Peiselulli ✓
Richard ✓
enthusi ✓
errazking ✓

Volume #1

3,27€

CONTENTS

Proofreading and improvements by Marc Walters
Cover illustrations by Lucia Piccin
Book layout by Stefano Tognon

Regame 64: It is never too late for a Commodore 64 (volume #1)
Stefano Tognon

Self-published online on 31/03/2017 with www.pixartprinting.it

PREFACE

This book is dedicated to all the old (and new) gamers out there for whom the Commodore 64 is still very much alive.

In this volume we explore some of the video games created in 2015 and 2016 and speak with their creators about the motivations driving them and the difficulties they encountered during development. We love the idea of giving creators a platform beyond their games on which to delight and inform their audience - and it is for that purpose we made this book.

We take a look at "Caren and the Tangled Tentacles", a point-and-click graphic adventure; "Let's Invade!", a new extended clone of Space Invaders; and "Donkey Kong (2016)", a faithful port of the arcade classic!

Although most of the book is dedicated to games, we also give space to some of the best 8-bit graphic masterpieces. This book showcases entries in the graphics contests at the X'2016 and Syntax 2016 demo parties.

We also examine a recent (im)possible project: a C64 port of the recent multi-platform game, "Limbo"!

This first volume has 40 pages. The next will be expanded to 44 pages if we can avoid increasing the postage costs.

The final chapter is dedicated to the making of this book.

You can reach the authors of this book at:
Website: regame.altervista.org
Email: regame@altervista.org
For information about IceTeam and their other productions:
Website: iceteam.altervista.org
Email: iceteam@altervista.org
Facebook: www.facebook.com/IceTeamIt

Caren and the Tangled Tentacles

Genre	Adventure
Released	PriorArt
Code	enthusi
Graphics	Veto
Music	Jammer
Year	2016

The LucasFilm games division was famous for their point-and-click graphic adventure games based on the **SCUMM** games engine. These include the commercial and critical hits *Maniac Mansion* (1987), *Zak McKracken and the Alien Mindbenders* (1988), *Indiana Jones and the Last Crusade: The Graphic Adventure* (1989), *LOOM* (1990) and *The Secret of Monkey Island* (1990). The first two games appeared on the Commodore 64, and the later games on Amiga and MS-DOS.

In those games you use a joystick or mouse-driven point-and-click User Interface to control an on-screen character and interact with objects and other characters to solve a series of puzzles to complete the game.

I'm sure that many of you will be thrilled to have a new adventure game on the C64 like the LucasFilm ones and this is not a dream, but a reality!

Usb key with CRT image of the game

Near the end of 2015 the development group PriorArt entered their point-and-click adventure "**Caren and the Tangled Tentacles**" in the Adventure category of the *Forum64 Game Competition 2015*. They won!

The game was coded by **enthusi (Martin Wendt)** with graphic by **Veto (Oliver Lindau)** and music by **Jammer (Kamil Wolnikowski)**. Their game submission was version 1.0. It was later extended and released as version 1.1, followed by version 1.3 which was a special limited edition for which just 100 cartridges were produced by *Retro Gamer CD (RGCD)*.

The cartridge-based release was a stretch goal of the *C64 Visual Compendium*'s Kickstarter campaign. The other backers received a USB pen drive the size and shape of a credit card. Its cover features a beautiful illustration of Caren amidst swirling clouds, and the flipside has a stylised cassette.

The USB drive contains a CRT (C64 cartridge) image file of the game plus PC, Linux and Mac versions of the popular C64 emulator VICE.

After the game loads you are presented with an initial menu from which you can choose the following actions:

- read the graphical game manual
- set the in-game language to English or German
- load a previously saved game

The main game screen is split into three

Don't miss the graphical manual

distinct areas: the topmost reserved for speech text, a window in which the game locations are displayed, and a graphical control panel. When a game is not in progress an attract mode plays, featuring a nicely animated Caren introducing the people involved in the game's development.

When you start a new game you are

I like the Caren font!

transported to a colourful, wonderful world in which you can practice using the innovative, entirely joystick-driven, control system. The joystick positions the targeting reticle within the location window, a brief button-press moves Caren to the reticle, and a longer press activates the action panel in which location and inventory objects can be manipulated.

"What is that sound? The phone! . . . Nooo, it's woken me from my pleasant dreams." Your day has begun with a telephone call - you need to go to the chemical laboratory.

But not before your morning ablutions! Caren can use the bathroom, cook breakfast, watch television, play - yes, actually play - a classic Pong video game. She can

Dream ... it is colored!

buy goods at a local store, then take a bus to the laboratory where she will promptly discover a corpse - and a mystery.

Okay, enough with the spoilers! We shall now delve under the hood to examine the game engine.

First of all, this is a true multitasking engine that supports 32 threads. This allows

It is better to make the bed!

I'm filed! And now?

simultaneous interactions with non-player characters (NPCs) and objects, plus cued animations such as the elevator doors that open and close.

Second, the character masking works perfectly. The locations have depth on the z-axis, which means Caren may appear in-front or behind objects. However, software clipping (which is computationally expensive) was not used. Instead, the developers chose a technique involving hardware sprite masking. This introduced colour limitations on the maskable objects, but the tradeoff allowed the game to run at a full 50hz screen refresh.

The game locations are displayed flick-screen, and the clean and simple graphics demonstrate artistic skill and attention to detail. The graphics, in this author's opinion, surpass that of Zak McKracken. For example, Caren's lips are accurately animated when she speaks. Sprite animations are fluid and the pathfinding works perfectly - Caren automatically navigates around obstacles to reach the indicated position.

The music and sound effects are many and varied, although it should be noted they're written primarily for the 8580 SID chip. Sound is cued to the scenes and events. A good, and somewhat shocking, example of this can be heard upon first entering the laboratory in which the victim's body lies.

Gamers well-acquainted with point-and-click adventures might find the game easy to solve and perhaps too short. However, I think that even those players will enjoy the game's clever puzzles and innate charm. In my opinion, this game is one that shouldn't be missed.

I'm playing PONG! wow

No, no, no. He is dead!

Creator's corner: code with enthusi

When I saw Caren and the Tangled Tentacles for the first time I was instantly reminded of The Day of the Tentacle - but perhaps the similarities are coincidental. So, can you describe how this adventure game was born?

Usually I work on several projects at a time, like so many coders do - often only to the point that I am satisfied with the effect or engine. Sometimes those get re-used at a later point or, more likely, they will be redone from scratch when needed. While that might sound incredibly unproductive, it is what makes all the fun in this hobby. Coding challenges and obstacles and refining routines with every iteration. And of course when I work together with graphic artists or musicians we aim at a polished complete game or demo in the end. Such as '*Assembloids*' or '*Not even Human*' in the past (available via rgcd.co.uk or csdb.dk). With Veto I often discuss ideas and concepts, and we have already worked together on a number of occasions.

I spent many months and years reverse-engineering the SCUMM engine in Maniac Mansion and Zak McKracken for the C64. Well, the script-running engine was later called SPUTM. I was (and still am) amazed at what people came up with out of nowhere in the late 80s. I personally consider the point-and-click adventure with an actual character on screen to be the holy grail of 8-bit gaming. There is a reason why there are so darn few games of that genre on the C64 even though it had a pretty huge impact on gaming. Then in 2014 a Gamecoding compo was launched in the German forum64.de

Its goal was to code an adventure or RPG game from scratch within one year. I immediately knew that I wanted exactly that: a deadline to actually get things done and a wonderful opportunity to start with it. I believe I contacted Oliver 'Veto' Lindau the very same day, as it was clear that I would only want to do that with him. Firstly, because I knew we had very similar taste and understanding in games. Secondly, because I already had ideas for non-standard graphics tricks that I knew not many would be capable of. And thirdly, because his graphics simply rock and, while there are other great artists around on the C64, I know of no one who could pull off such a beauty in char mode with that amount of detail. In the past I always had fun working with him - which I guess is, in the end, the most important ingredient to any project, particularly a long lasting one such as this. We agreed within minutes on the rough setting of the game: detective style, 1980s, female character. A total cliche in a way, yet something very new to the 8-bit world.

We also wanted it to be a bit more serious than many of the more recent adventures for modern platforms. Many days of discussion and chat followed. In fact, we have chatted on an almost daily basis ever since the start of the project. It is a quite awesome process and, I believe, very different than what you would expect from larger, professional teams. We each threw in many ideas - I guess both of us had a long lasting urge to do at least one graphic adventure. Our notes from that time read pretty straight forward. Point-and-click, perspective such as in Zak McKracken, no scrolling, multicolor charset, special mode for animation, items as pictures at the bottom area, three sprites per character, and so on and so on.

You have implemented a perfect system for the action/inventory; you do not need to use the keyboard to choose actions to perform, but instead the joystick button is held for different durations to achieve a variety of operations. How did you arrive at the idea for this system?

That idea surfaced rather early in the overall process as we both deemed the control scheme of such a game very important. We did not want to use the pointer-dragging verb-interaction style, nor did we want to render it as a mere arcade-style game where you simply clicked on things and the character would do the right thing on its own. We iterated a lot over how this can be handled. Major influences were the later Sierra games that cycled through actions on the spot in the screen and the Action menu in Full Throttle. At the same time we knew we did not want an onscreen item system with bags full of bags, etc. The player should see what he is carrying and the interaction should feel natural. We ended up with the concept of three actions controlled via a separate Action panel that can be entered by a long button press on the joystick - no keyboard control required.

Ron Gilbert not hating Caren

Veto had a vision of that early on and his very first mock-ups of the panel are already pretty close to what we have now. The competition version featured the actions *Use*, *Look At* and *Use With* to interact with an inventory item. A double-click of the button launched either *Use* or *Look At* depending on the object. Since then we've refined that further; we removed the double click and added a fourth action to quick-exit a room whenever the exit is accessible (i.e., stairs, open doors and roads).

It was of great help that Veto was able to demo a super-early rendition of the engine at *Gamescom 2015*. Seeing casual gamers as well as pros of all ages and sexes (including Ron Gilbert) play our game was a great motivation and also provided extremely helpful feedback. We did the same with the current game at this year's Gamescom which, along with all the helpful feedback we got from players of the game via Emails and forum posts, led to further small improvements.

Implementing a multitasking system with the 6510 CPU is not an easy task - but in the game it works very well. How did you implement it? Also, the game has 32 threads running - can you briefly describe each thread?

I knew that on a C64 the game would hit two major challenges: the graphics engine dealing with all the graphics for rooms, items animation and characters; and, of course, the engine running the game-logic itself. Any restrictions in that game-logic engine might later limit what we can do - but at the same time it has to be as efficient as possible. So: a fast, small, flexible, yet generic engine. When you code a game in BASIC you inevitably end up with endless rows of IF THEN statements. That is fine for text adventures, but for a game with live interaction you need another approach.

I was always amazed at the things going on the background in Maniac Mansion and Zak McKracken, such as the ticking clocks or moving escalators at the airports. That multitasking system was, in

fact, the first thing I coded for Caren. At its core it checks thirty-two 16-bit timers every frame and decreases them if they are still running. Once a timer ends the following information is read out from its corresponding task-slot: the amount of frames for the next call (if any), the 16-bit address of the routine to call, and three additional bytes that go into the three CPU registers before calling the given routine.

This is the sourcecode for task-slots occupied by the engine:

```
.byte 1,5,<_flash_pointer,
          >_flash_pointer,1,2,3
.byte 1,3,<_update_actions,
          >_update_actions,1,2,3
.byte 1,19,<_update_mimik,
          >_update_mimik,1,2,3
.byte 1,23,<_poll_keys,>_poll_keys,1,2,3
```

So, every 5 frames the pointer changes color, every 3 frames the engine checks if the pointer hovers over an item or object and highlights the available actions accordingly, every 19 frames the actor's mimic is updated and four times a second the keys are polled to see if someone has tried to load/save a game or change the game's language. All these do not require any additional external information and run on their own. If, for example, the game-script for a room contains a line like this:

```
say "Nun ist es besser.",0,"That's better
now.",0
```

This is inserted as a task which carries the address of the line of text in registers X/Y, the talker's ID in register A and a reference to the game routine *_talk*. The timer is set to launch *_talk* in the very next frame. The routine *_talk* replaces itself in the multitasking flow by a function that gets called repeatedly; it stores the text display area, prints the text and parses the spoken text letter by letter to animate Caren's mouth. When the task-slot gets called and parses the last letter it starts a wait-loop and finally restores the graphics of the text display area.

The game never reaches 32 simultaneous actions but in that event the engine will balance priorities. The first slots have low priority, the last slots have high priority. The engine checks the slots starting with the last one. For example, a door animation might be assigned slot #31, so this will be done first.

This is important as Caren runs at full 50 frames per second and uses no double buffering. The multitasking system starts when the raster beam reaches the end of the room graphics. The tasks that affect on-screen graphics are dealt with first - completed before the raster beam begins drawing the visible screen of the next frame. Computing pathfinding steps is a low priority routine that can be executed during display. If raster time runs short, the next slots, such as for key-press polling, will be skipped that frame to prevent flickering sprites and other graphic glitches.

We will be smart enough not to design a room with 10 large doors that can all be animated in parallel ;-). But if we did, the engine would decide to delay the animations from say door 5 on by one frame. This is also why *update_actions* and *flash_pointer* above occur in prime number intervals; so they overlap as little as possible.

I know you're following the Ron Gilbert diary of his new adventure game, Thimbleweed Park. Ron is one of the fathers of adventure games on the C64, and the creator of the SCUMM scripting engine. What can you tell us about your

scripting engine?

Making adventures without a scripting engine can be very problematic. Yes, you can try the approach of hard-coded IF-THEN loops for each individual room, but this will neither be much fun, nor very flexible. Even after the initial competition version the core scripting engine was basically done, which made adding new rooms and individual features much more feasible. While I am somewhat familiar with the format of SCUMM, I chose a very different approach for several reasons.

SCUMM generates some sort of byte code which the game engine then interprets in real time. The game script reads *"say 'hello'"*, the byte code then tokenizes say into its own single-byte opcode. The scummvm team did an amazing job reversing much of this concept, and every now and then Ron Gilbert mentions historic examples of it on grumpygamer.com or pagetable.com (see http://www.pagetable.com/?p=603, by me).

This approach has the smallest memory footprint with regard to the individual scripts that need to be loaded for every room, but on the other hand it requires a much larger engine as it has to implement even the simplest things such as addition, subtraction and addressing. At some point I decided to got for a scripting language that gets directly compiled or rather translated into 6510 assembler. Every game logic script in Caren is executed as raw assembler code. The scripting language is rather simple (since I had to write my own 6510-Compiler for it ;-)

Consider the following script:

```
walk2obj +8 shelfmiddleleft
keep_walking
break for 20 frames
say "Dann untersuche ich mal.",0,"Ok,
let's analyse the sample.",0
```

It means:

- **Have Caren walk to a position 8 pixels right of the middle left shelf,**
- **Pause the script until she arrives,**
- **When she arrives, wait 20 frames,**
- **Say the text.**

It compiles to:

```
        ldx #35
        ldy #60
        jsr _set_walk2xy

wait_till_talking_ends
        ldx #<(*+9)
        ldy #>(*+7)
        lda #20 ;in that many frames
        jmp _break
        lda game_actor_walking
        eor #$ff ;ff=not walking
        bne wait_till_talking_ends

        ldx #<data041
        ldy #>data041
        lda #15
        jsr _talk
        bvc data_end041
data041
.asc  "Dann  untersuche  ich  mal.",0,"Ok,
let's analyse the sample.",0 ,0
data_end041
```

The XY coordinates of *walk2obj* are directly computed during compilation time based on the given objects and their positions in the script. The engine chooses if the first part (German) or second part of the text is printed. Both languages reside in memory at the same time! The most helpful game routine is probably *_break*. It enters a new task into the system which jumps to the address after the break point (the (*+9) and (*+7) are 16-bit pointers) in the given number of frames and ends the current task. So at any *_break* the script becomes sort of parallel. The scripting language currently understands the following commands:

```
if, call, set, end, say, play, animate,
setcurrent, break, walk2xy, walk2obj,
walk2npc, pickup, dro[,inc, dec, goto,
launch, loadroom, ifown, ifnotown, keep-
_listening, keep_walking, enable_exit,
disable_exit, reconfig_area, recon-
fig_object, place_caren, loaddata,
```

```
loadsid, hear, ifused, ifnotused, en-
able_passive, disable_passive, exchange,
announce
```

Some of them have optional parameters such as the +/- shift of pixels for the walk2 command. The reconfig commands are particularly powerful. This is an actual line in the script:

```
if shelfmiddleleft_glasfull
  reconfig_object shelfmiddleleft
  usewith _analyseglas
```

It means IF there is a glass in the middle shelf which is full THEN the function to call when Caren uses it with an object is set to *_analyseglas*; its default is the dummy for 'none'. The disadvantage of compiling into 6510 assembly is the size of the individual scripts. Things occurring often, such as animation, are implemented as a global routine of the engine. There are two main advantages though:

The first is its flexibility; my compiler accepts single assembly statements as well as full blocks of assembly in the script. For example, the routine to fade in Caren's bathroom did not require a lengthy extension of the core engine but instead is directly written in assembly inside the script. The same applies for special cases that deviate from the standard procedure in the engine.

The other advantage is that I can now link several (in fact, ALL) rooms at source level; all core engine functions are rendered globally. Also, each script can export variables and flags to make them accessible from other rooms. These variables are kept in memory during the game. It only costs a few KBs and has a huge impact on the gameplay. Any drawer or door that you open or close will maintain its state over the course of the game. This seems trivial, but it's not common for games on the C64 or any other 8-bit machine.

It is less an issue of memory and more one of organizing all the data. Any change in any room or the engine potentially requires me to recompile and reassemble every single room - which is reasonably fast on a cross-plattform PC today, but unthinkable for a disk based development system thirty years ago. Changes in the engine are particularly tedious as they impact all rooms.

You used a new approach for the clipping of the Caren sprites when she moves behind foreground objects. Can you discuss this system and any advantage (or disadvantage) it has over software clipping?

The way we mask Caren to the scenery puts tight restrictions on all objects that were to be foreground or something Caren might walk behind. The approach is known in the demo scene as 'cookie cutter', and, to my knowledge, has never been applied in games before. It uses the fact that the eight hardware sprites are displayed in a strictly prioritised order and each can be flagged as 'foreground' or 'background' - which controls whether the VIC displays it in front of or behind bitmap (ie, non-sprite) graphics. This is unique to the Commodore 64.

Clipping example

Thus, a sprite can be displayed in front of another while still being, technically, behind the background. The net result is that the sprite with lower priority gets clipped by the mask-sprite and the bitmap shows through. So whenever part of Caren is supposed to be covered, we place a sprite with the proper mask-shape in *front* of her - but *behind* the background.

In the “Clipping Example” image you can see the sprite-mask layer in red, green and blue, along with the scene as it normally appears - with Caren behind the windows of the bus. You can also spot areas of hires pixels around the tires and luggage bins. There are almost twice as many sprites needed than are shown here since one is needed for transitions in between those that are shown. Whenever Caren walks in front of the bus all these sprites are disabled. It is not quite that simple though, since the background itself is 'half transparent' and the masking sprite shows through. In fact, two of the four bitmap colors per tile are transparent.

We win one color back by giving the mask sprite itself that very color. So every graphical element that might be in front of Caren can only have three colors. The global background color and the first global multicolor character-mode color are always transparent. Veto (the artist) can select one color for a character as well as the color for the mask-sprite. Using sprites as masks limits the size of maskable objects but the engine can (and does) generate masks dynamically. They are in hires mode and, where needed, expanded on one or both axes.

To be able to fully cover Caren, such as when she’s behind columns, the dynamic multiplexer re-uses Caren's head sprite for any mask below her shoulders. Two Y-expanded masks are always sufficient. There are screens where Caren walks behind wide objects such as a six-foot fence or the windows of a bus. As she walks there, the two masks are moved with her and their shapes adjusted (via sprite pointers) to match the object she is currently behind.

Using this somewhat unintuitive hardware masking technique saves a lot of CPU time compared to software-clipping. Software-clipping is one of the main reasons why the scrolling in Maniac Mansion and Zak McKracken is slow and the characters move horizontally in character-wide steps (you probably never noticed that until now, though. Sorry!). Hardware masking enables our game to run at 50 frames-per-second (FPS) on PAL systems (60 on NTSC) without buffering either sprites or graphics.

However, there are some disadvantages. Using precious sprites for masking further reduces the amount of “free” sprites per room. Only one non-player-character (NPC) can be masked - partially. The multiplexor reuses its head sprite to cover ONE of the two persons on-screen. For example, we can have two characters on-

Guy on the bench

screen with a fence in-between.

But that fence must be lower than the head of the person behind because we can not cover the head in the back independently from the foreground. So a bench, garden fence, dust bin etc is fine. That particular situation doesn't arise in the released version but the engine supports it. Also available are masks for BOTH (the global mask), such as a pole or sign.

AS a result, the whole sprite set-up is limited to two freely moving characters at a time! But veto has already demonstrated in our current development version that NPCs can be rendered convincingly as character graphics (see the guy on the bench). Any software-based masking on that scale would slow down extremely with more than two 3-sprite-high characters, though.

What algorithm did you use for finding the path for Caren to move to where the player clicks on the map? That feature has to be fast and reliable - and it seems to be.

That was indeed a big one and took lots of coding. Yet, I knew from the beginning that it was possible at least, as Maniac Mansion and, even more so, Zak McKracken, had already done a somewhat decent job of it. The early SCUMM games build up a walkable area consisting of several contiguous boxes. For Caren, I use that same approach. Up to sixteen boxes make up the walkable area.
I run Dijkstra's algorithm for finding the shortest path between any pair of boxes. This yields the number of the first box, *Box C*, to enter on your way from *Box A* to *Box B*. Once you are in *Box C*, the new task is to get from *Box C* to *Box B* via the next box, and so on, until you reach the destination box.

Restricting myself to 16 boxes allows for a beautiful 16x16 matrix that fits in one single memory page of the C64 and thus can be addressed indirectly. You can imagine a matrix with FROM as rows and TO as columns. Each entry tells the engine which box to aim for next. Now this would be simple if Caren would jump from box center to box center - but she doesn't.

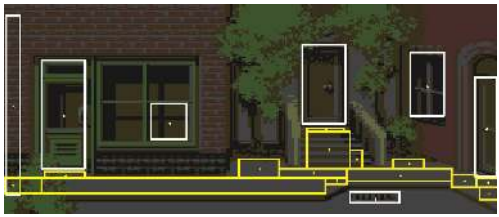
She will likely not start in the center of any box, nor will the destination sit in the center of any box. The shortest path might not even cross any box center for that matter. What happens is the following:
•Caren aims for the closest point on the side of the next box that touches the box she currently stands in.
•Once reaching that point on the overlapping sides she again aims for the closest point on the closest side of the next box until she reaches the side of the final box - which lets her target the destination directly. Any rectanglar box is convex, so any point within a box can be reached directly from any point on any side.

It is quite nasty to actually implement this in a practical way, such that she never leaves a box even when walking along one of its sides, and that she properly carries out the transition from one to the other box when aiming for the sides. One costly CPU problem is to determine which of the four sides of a given box is actually the closest to a given position. To select the closest you would have to check all four sides, find the closest point on each of them and then compare the distances to the given position. This can

be sped up since you can omit any square roots - you are not interested in the actual distance, but merely the order of distances. Yet, this takes a considerable amount of time in the MM and ZMK game engines. For Caren, I decided to precalculate even that. Any matrix entry has a unique SOURCE and DESTINATION box, and carries a single number for the next box.

It was handy to have 16 boxes for a 16x16 matrix but it is even more handy now as I only need 4 bits of the matrix entry to identify the next box. I use the upper bits to give information on the side from which to enter the coming box (as well as information if it is currently being blocked, i.e. for closed doors). So instead of four computations, I now only need one - which still takes a notable amount of raster time.

So you find the closest point on a vertical or horizontal line and then move to that. The 'move' part again is a bit tricky. Usually you would apply some Bresenham algorithm but even that felt too long for this free directional approach. Instead, I derive a fractional *dx* (direction-x) and *dy* (direction-y) step in 8.8 format. To get there I take the absolute *dx*256* and *dy*256* to the target as 16-bit words and divide them until their sum gets below a certain threshold. At the same time, I double a steps-variable and thus end up with the amount of steps Caren needs to take of the size *dx*, *dy* to reach her destination. This is very fast but leads to slightly different walking speeds for Caren depending on the length of her current path. The animation takes this into account though, and in the end it works out quite well.



Xfig: yellow - walking boxes, white - trigger zones

Deriving the proper target position is not trivial either, once you think about it. You generally do NOT want Caren to walk to the pointer, but to the point BELOW the pointer in the walkable area. Walking to a framed picture on the wall means walking to a point some 70 pixels below the actual pointer. So every time you press and release the button to guide Caren somewhere, the engine checks if the pointer already is in a walkable area, then it takes the pointer's coordinates as a target. It corrects for the offsets of the sprite in the upper left to the pointer's center. The same applies for Caren, who has an offset for the position between her feet.

If the pointer is outside a walkable area it first checks all 16 boxes to find the one closest to, but below, the pointer. If that fails it checks above the pointer (which happens, though less often, such as when pointing at a road but being limited to the walkable sidewalk above it). Finding anything that is 'closest' always requires a check of all boxes, although you can optimize it by assuming most walkable areas are aligned horizontally; so you check those boundaries first, and so on.
I drew all walking-boxes in Xfig, an open-source vector graphics program written in 1985.

What development environments and programs did you use to develop the game? How many times did it have to be rewritten?

Actually, I never restarted from scratch during the process. The previously mentioned routines for walking and the multitasking system were done independently and then merged at some point. For some time now I've been using a versioning system that helps me keep track of things - as well as the current status, since I code on several different machines. The assembler I use is *Floodgap's XA* assembler, which comes with a suitable disassembler and uses a pretty straightforward syntax - which I prefer. The only things I use beyond bare opcode translation are *#defines* and local labels. The whole assembly is organized via *Makefile(s)* and many python scripts I've written for individual purposes. Most notable of these scripts is the converter for the room graphics and animations, but they also include the script compiler, file system generator and the box-matrix pathfinding routines. I follow the KISS (Keep It Simple, Stupid!) approach: have individual tools for individual tasks.

The project's Tools folder contains 38 different python scripts at the moment (some are in other directories). Some scripts are really simple, such as *set_version.py* which takes care of my internal versioning of the engine. I like *avoid_mayhem.py*, which runs consistency checks against the scripts, and *grapher.py*, which automatically generates a giant chart connecting all rooms according to their accessibility from each other. Each room is represented by its main graphics and a colored frame representing the music (or sound-effects catalog) used in that screen as well as all the items you can get or lose. It generates DOT code for utilities such as *Graphviz*. It's little things like that generate a lot of fun - and save lots of time, too :).

I do all the development on linux systems, mostly running on a rooted ARM Chromebook or Raspberry Pis. A simple text editor with personalised syntax highlighting such as *Kate* or sometimes even *mcedit* from *Midnight Commander*. *Mercurial/Bitbucket* is used for versioning. *Python* is used, of course. For a packer I used *Doynamite 1.1* from *Bitbreaker* - although right now most files go unpacked on the *Easyflash* with a loader optimized for speed. At some point even the 1 MB of the *Easyflash* cart will not suffice for raw data, so some selective packing will come back into play.

These days it is very easy to feel superior when doing cross-development for 8-bit platforms. The tools available have a huge impact on what can be achieved. Also, many of the chosen solutions to coding challenges are based on what has been available and continually refined over a long period of time. Yet, one should never underestimate the skills early game coders (well, some at least) had in the 80s. After all, they did it for a living and were coding in 6502 assembly all the time. People in 2016 are not smarter than 30 years ago. Take away the tools and the internet then things will start falling apart for most of us. While the demo scene reaches new heights on almost an

The game title screen

annual basis, the code quality as such does not change too much.

Many achievements are based on a better understanding of the hardware (including illegal opcodes), but not so much on progress in the art of 6502 assembly. In the last few years hobbyists have simply started to put way more time and energy into their projects - which is why so many new great demos and, occasionally, games surface. One of the reasons why things worked out so well for Caren is that, for some time, I have worked on subsets of the challenges involved over and over again, read other people's code and reversed interesting game routines. It helps to have seen many games and implementations to at least know what to avoid. Also, Veto is pretty knowledgeable when it comes to modern games as well.

How did you feel about working with Veto (graphics artist) and Jammer (sid musician)?

I still work with both of them and it is always a great joy. Without Veto and me working so well as a team this game would not have surfaced at all (as happens with so many other game projects these days). From Day One we managed to make decisions on the engine or game design very quickly - which is wonderful. We each respect each other's skills and have a good idea what to ask for, and what not to. There were some ideas I really wanted implemented, such as the animation approach, even though I knew it would be an extra pain for Veto. There were things Veto had a strong opinion about, such as the asynchronous sprite animation of arms and legs and the head bumping in between. But we are also fine with disagreeing over some ideas - which, in the end, helped the game a lot, I am sure.

Jammer joined at a rather late stage before the original competition deadline, and several tunes and sound effects were changed or even implemented in, quite literally, the last few minutes - which was quite a thrill. When Jammer agreed to work on Caren and also join PriorArt there were some choices to be made; Jammer didn't like the idea of simple sound effects on a third voice replacing part of the running music. Thinking through some possible variations and alternatives we decided to follow an all-or-nothing approach - with Jammer being the Sound Director. Each room has either a wonderful three-voice SID track or sound effects implemented as complete sub-tunes, thus having available the entire capacity of the SID.

That required a major effort from Jammer since each SFX is a little tune in its own right, but it was a decision that led to, in my opinion, the best sound effects you have ever heard in a game. The tunes themselves are outstanding and Jammer made them specific to certain areas of the game. The sewers have their own mood in contrast to the friendly street themes and the dramatic climax near the end of Caren 1.1.

For all three aspects of the game - engine/scripting, graphics/animation and music/sounds - we tried something new, which I think worked out quite well but only because the three of us cooperate so well. Jammer has added a lot since version 1.1; and once again, while I asked for certain directions of tunes in a few cases, Jammer had his own vision of how things could play along nicely, such as the con-

tinuous playing of a tune across several screens of the same scenery, such as the street or sewers. A nice example is the pong game you can play within Caren.

I had stumbled upon a detailed description of each of the sound effects of the original arcade game, which included waveforms and exact frequencies. I asked Jammer to implement all of them, just for the fun of it. That was at a time when things had already became pretty tight. Jammer not only composed all those effects but he also added distinct filtering to mimic the characteristic sound of common TV speakers of the 80s. That is a prime example of what I meant about everyone contributing his own OCD to the project ;-)

For several scenes we discussed very precise aspects of scripting, audio and animation. The music has quite some depth to it as well. You may recognize some small references to known pieces of music or to the action on screen. For example, if you pay attention to the tune in the second scene you'll notice the growing presence of the phone's ringing interwoven with Caren's dream experience.

Caran on cartridge!

The Kickstart (limited edition) version produced for cartridge gives a boost to the game and many are asking for a follow up. What can you say about this?

After the competition we spent some weeks fixing, polishing and proofreading for a stable release of v1.1. That one is freely available. You can get it from here: http://csdb.dk/release/?id=141659

The "Kickstart" version is a greatly enhanced limited version produced for the backers of the book *Commodore 64 - A Visual Compendium: 2nd edition.* Each backer received it on a USB drive, bundled with emulators. There were also 100 physical boxes produced, with the game on a dedicated cartridge designed by Siem Appelman. While that limited version included many new scenes, the overall plot stayed the same and it ends with a cliffhanger - and indeed, we will carry on.

We are in the process of creating an even larger adventure for Caren. Changes applied to the game so far will contribute to the first chapter of the new game; by this, I mean that it will be one large game but, as with Monkey Island, the game will consist of distinct episodes.

We designed a small city map in which all buildings and streets will eventually be accessible. The player should be able to move around freely (well, of course you can't just enter any house any time :)). This will also allow us to implement a few more characters into the game, as we've done with the version used for the second Visual Compendium book's Kickstarter campaign.

Right now we are in the process of polishing and extending the engine. Some existing rooms need upgrades as well,

even though all have worked well so far. From feedback, our own experience and simply watching people playing the game we have obtained some ideas on how to render the gameplay even smoother.

Players at Gamescom

There will be no double-click required any more. Instead, the quick-exit option will become part of the action-panel. We cannot provide any release date as this depends on many things: family, work, health... We're aiming for a 2017 release, though. The game has became huge already and will grow considerably - which is why this will not be released as a disk-based game. The exclusive book-backer version was distributed on 1 MB carts and, obviously, the final game will not be smaller.

Also, we will continue to support German and English versions of the game :). By now, the game logic scripts have surpassed 12000 lines and there are more than 650 lines of text and 13 SID tunes, with more than 40 additional sound effects.

Have you any special information or consideration to share with us about the game?

We presented the game at the Gamescom 2016 in Cologne. It is the world's largest games exhibition, with around 350,000 visitors. Although not everyone played the game we were quite delighted to see the amount of people who stopped by and took a turn on our game. Young (we provided a stool so they could see the screen better), old ("See kids! Those were the days!"), men and women. Despite a point-and-click adventure game being less than ideal for passers-by to learn and play (unlike casual shoot-em-ups), our stand was almost always occupied (and so were we!). We were amazed at the large variety of players: grandpas showing their nephews that 'old machine', people who could not believe that what they saw ran on their childhood computer, and folk who were considerably younger than the hardware they were playing on.

To us, it was also a great opportunity to finally meet up with Robert Megone, who helped a lot with proof-reading, translation in general and, of course, play testing :). Robert is the lead tester of Ron's current project ThimbleWeed Park and is also involved in ScummVM. He approached us via Facebook and IRC (yes, really! :)) to offer his invaluable help. Thanks again, Rob! We had a great time and kept on discussing ideas and concepts for Caren's future. I consider him a very valuable member of our (and any other) team.
http://www.robertmegone.com/

Classic-Computing in Nordhorn

Meeting him and other adventure game maniacs at events such as the Adventure Treff party was extremely motivational - although, frankly, we never lacked motivation during the almost

two years of development. A few weeks later we had the opportunity to show the game to the retro-hardware-focused community at Classic-Computing in Nordhorn, Germany. While there we had more time to have detailed discussions with interested people, and to brainstorm ideas about what to include in the large game project that this has now become.

Veto, Robert, Enthusi at Gamescom

It feels very good to see people play and enjoy the game. Although *seemingly* trivial, my experience is that it is not; far too many games are merely being collected and owned - but never played. A further surprise was the multi-page article featuring new C64 games in *Gamestar,* Germany's largest game magazine, which featured *Caren and the Tangled Tentacles* along with the other major C64 titles from this year's Gamescom, *Sam's Journey* and *Tiger Claw.* The article, in German, can be read online here: http://www.gamestar.de/specials/spiele/3301527/gamescom_2016.html .

Veto at Meteoriks

All of the featured C64 games are unique creations - not ports or de-makes of popular originals. I believe it's that spirit which captures the attention of the players. A single screenshot of *Super Mario* or *Monkey Island* on C64 might attract ten times more people than something completely new - but would such a title actually sustain people's interest?

Would you play *Super Mario* or *Civilization* on the C64 if it existed on other systems? To me, it's those titles that embrace the C64's restrictions and work with them that ultimately stand out. A game is so much more than just code, graphics and music! At the end of the day none of us is a professional game designer; however, no one was in the early days, either. It is probably the spirit within the team that dictates whether a game is to be enjoyable or a mere technical demo.

We were stunned when our game won the "*4Players Best Scene-Game Award 2015*", the "*Forum64 Game Competition 2015*" and the 'Interactive' category of the cross-platform "*Meteoriks Scene Award 2016*". The four of us truly enjoyed planning and working on this game - and there is nothing more satisfying than the fact that people enjoyed playing it.

Lemming recently made a short clip of the version received by the bookbackers: https://www.youtube.com/watch?v=7lY5-wg_LHE

Let's Invade!

Genre	Arcade
Released	The New Dimension
Code	Richard Bayliss
Graphics	Richard Bayliss Igor Errazking
Music	Richard Bayliss
Year	2016

"4th December 2077: Earth has successfully made contact with alien life forms. Luckily, most of them are friendly and welcoming. Humans and aliens are working together on several planets of the Solar System to bring about a peaceful galaxy and a better worlds for all. One year later, an unidentified giant mother ship approaches planet Earth. Scientists work hard to establish communication with the giant ship. The translation of the strange alien signal is successful... but it brings bad news. The new visitors have not come to Earth to help out. They are very greedy and want to invade all planets. Planet Earth is at the top of their list. A message from Planet Earth is sent back to the alien leader: Earth will fight back! The mother ship beams out a psychedelic vortex

A pre-title animated screen

field, and spits out hordes of aliens... ready for the invasion. Planet Earth beams mobile military bases, and prepares to do battle with the aliens inside the vortex field."

Back in the old days I would buy a game cassette and, while the game was loading, I'd read the story (much like the one above) printed on the cover-slip and wonder what I'd find when the program started. So, after all that, what can we expect from *Let's Invade!*, the new game by Richard Bayliss of programming group The New Dimension?

If you were thinking "Just another Space Invaders clone" then you are a long way from reality. *Let's Invade!* is different, even if it is based on the classic *Space Invaders*.

Once loaded, the first thing you are presented with is an introductory screen similar to the "intro" used on cracked

It's an alien fire storm!

games. This one has a TND logo, a scrolling text message and music.

Before I say another word you should know that I cannot be objective when speaking about Richard's music as I consider him as Matt Gray's successor - with

a dance style that I could listen to for hours! I'm also pleased that Richard created the music for my own game, *Little Sara Sister 1.5* ;).

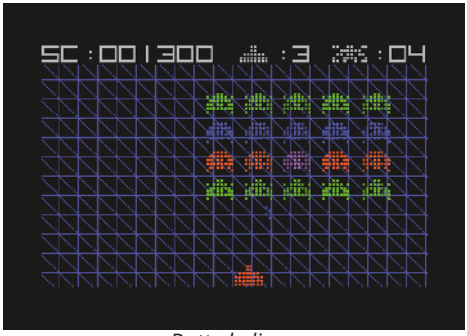
I listened to the intro music for 30 minutes then pressed a key to continue. Next appeared the game's main introductory screen, accompanied by a "techno" style music track. The title picture by Igor Errazking depicts a menacing, blocky invader silhouetted against a fiery explosion - which I find very appropriate for this game.

Despite the blinking "Press Space" prompt which eventually appeared in the upper right-hand corner I stayed on this screen for another 30 minutes just to listen to this hypnotic track. Damn you, Richard! :)

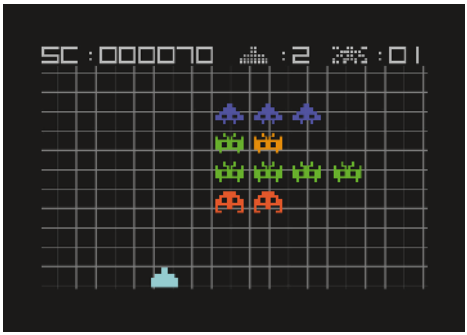
Once we continue (or escape) the main intro we encounter a warning message with an option to disable game features which might trigger epileptic seizures.

A perfect title screen

After this we reach the game's titlescreen. In my opinion this is one of the best ever created; it is somewhat reminiscent of "*Boulderdash*" - scrolling background inside the text, an animated alien, blinking stars, scrolling text message, and (once again) another 30-minute "don't-press-the-button-just-listen-to-this-damn-music" track that begins as innocuous alien-like burblings before launching into a full-frontal audial assault, Richard-style.



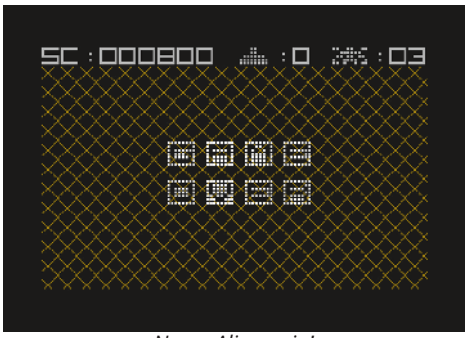
Dotted aliens



Solid aliens

On the titlescreen you use the joystick to choose the in-game graphics to be either blocky solid-fill or large, transparently bordered pixels; and to select whether music or sound effects will be played. The game begins with a static screen in which 25 aliens arrayed in five rows must be destroyed, just like in *Space Invaders*. However, there is a significant difference compared to the original arcade game: there is a scrolling background that flashes to the sound of the drumbeat in a psychedelic manner. Wow!

The game increases in difficulty from level to level. The well-written PDF



Nooo. Aliens win!

manual describes the changes encountered. One example being that in lower levels an alien can be killed with a single shot - but it takes two shots in later levels. Also, the alien bullets are faster in advanced levels. Fortunately, you receive helpful "power-ups" by killing orange aliens, and extra lives for destroying purple aliens.

The game totals 40 levels in addition to a proper animated finale (I cannot say more otherwise I'll spoil the surprise!).

Several more excellent tunes, such as the one on the animated highscore table, have meant that I spent hours just listening to music while reviewing this game! Thanks Richard :).

Let's Invade! is an example of how a game should be: it is complete in all aspects, has innovative gameplay that respectfully extends the original concept, and if the in-game "trance" music is not to your taste you can replace it with sound effects.

Creator's corner: 360° with Richard

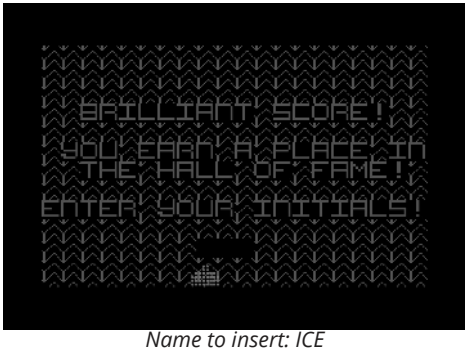
Space Invaders is a classic which has inspired the creation of many clones on the Commodore. How did you get the idea for this innovative version?

I was a big fan of the original arcade classic, and as a C64 musician, as well as a programmer, I saw a few C64 Space Invaders games. I always had a dream to make one for the C64, but of course it had to be something slightly different :). I had a modern day idea for a C64 Space Invaders game: a psychedelic stylish twist, with in-game background animation and my traditional 8-bit SID trance music.

What equipment and programs did you use for making the code, graphics and music of the game and how much time was needed to complete it?

I mostly used cross-platform utilities for the project.

Spritepad was used to create the game sprites. *Cuniforme* was used to make the game background and charset graphics. *D64 Editor* was used for importing/exporting program data from/to .d64s.



Name to insert: ICE

To program the game I used *Notepad++* with *KickAssembler* cross-assembler and *Exomizer* plugin. For making the trance music and in-game sound effects I used *Goat Tracker V2.27*.

Exomizer was used for the best possible

compression - without too long a waiting time for decrunching.

I used my *Tape Master Pro V3.0* utility to generate a tape master, and Martin Piper's custom IRQ disk turbo loader for playing music and displaying the picture while the game was loading.

Action Replay M/C monitor was used to move the load address of the disk version of the game to a specific area, for the IRQ disk loader.

How did you make stroboscopic effects synchronized with the music, and how difficult or easy was it to achieve?

It wasn't too much of a difficult task. I used Style's *Music Analyzer* to work out which position in the music's memory triggers the trance drum sound. I noted down the area which I should check for, then implemented a check subroutine which detected the value of the drum in-

This is a true HighScore!

struments (which were displayed in the music analyser). The flash effect was set inside a loop. After the flash effect table value reaches the end it gets reset to one before the last value. Every time the drum sound was detected the pointer which controlled the value of the flash was set

Takes some time to read this

to zero so that the flashing effect restarted.

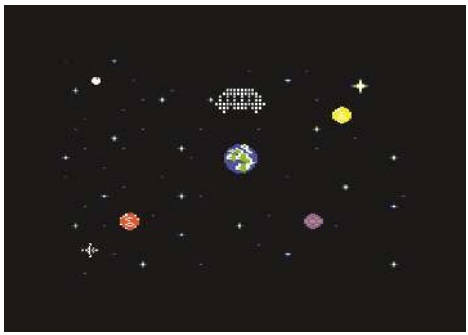
Are you satisfied with the increasing degree of difficulty you gave to the various levels and the power-up you inserted into the game that make it very different to the classic Space Invaders?

Yes. Although, personally, I have now learned I really should have made the game harder to play by increasing the actual speed of the alien bullets. The game was pretty much too easy to play ;). Also, I probably should have programmed my own sound effects player instead of using the built-in SFX player inside *Goat Tracker*. This was because of the delay before actually playing the sounds in the game. The faster aliens don't play sounds at all. Hmm.

You have added the possibility to change the alien graphics and to choose between music and sound effects. Are there any features you wanted to add but didn't?

I would have loved to have added a bitmap background into the game but, due to the amount of memory that I used, that wouldn't have been possible. Also, as mentioned above, I wish I had programmed my own sound effects and

made the game harder to play. Still, I was happy with the result overall :).



What the hell is this?

Is there a possibility for an improved version or a cartridge version of the game?

If there is a 16KB cartridge compo coming in the near future I might consider making a deluxe version of the game - which could feature multi-colour sprites, harder gameplay, and of course my very own SFX player. This would probably compensate for the sound bugs in my previous release.

Creator's corner: graphics with Errazking

How did you get the idea for the image? I find it very apocalyptic and appropriate for the game.

It's a recurring idea in my head. I never saw ships in *Space Invaders* (except the UFO at the top).They look like aliens to me and that's the way I think they would be destroyed.

I have to say that Bayliss did not ask me for the drawing. As soon as I saw his game I worked on the picture between my games - "et voila".

How did you go about its creation, and how much time did it take?

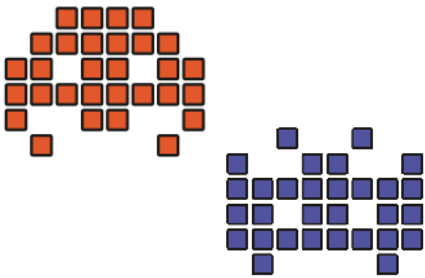
As I did it is a secret - no, not really - XD.

This version for the C64 is a revision of a version that I published in Miiverse ... if you see it then it will surprise you - it is the same.

Incidentally, the size of the drawings in Miiverse match those of the half-screen of the C64 ... To those who have access to Miiverse I recommend that you use it.

I imported the image to *Gimp* then added the background explosion and title then exported it to *Pixcen*. This type of conversion can be painful when you have to resort to even more tools to get it done. In this case I used *Ganged* to retouch the image to my taste and whim. Although general opinion suggests otherwise, drawing is not an overly difficult process (if you know how to) and doesn't take long to get a result. It usually takes me no more than two hours to draw a picture but there are always exceptions.

As a newcomer with only two years experience of the C64 I have to say that I am captivated by the complexity of something seemingly as simple as 8-bit graphics. Only those who have dedicated themselves to working with them are fully aware of the beauty and complexity that can be found within in such images.



Donkey Kong

Genre	Arcade
Released	Oxyron
Code	Peiselulli
Graphics	Veto
Music	Linus
Year	2016

Donkey Kong was one of the first arcade game ports I played on a Commodore 64. Nintendo released it many years ago - way back in 1983 - and it had been a real shock to learn the C64 was able to reproduce a game that previously we'd only ever encountered at local bars - housed in a bulky cabinet and playable only if we fed it a coin. Today, in hindsight, I reckon my initial shock of seeing *Donkey Kong* on a C64 would have been even greater had I encountered the new port that coding group Oxyron released earlier this year!

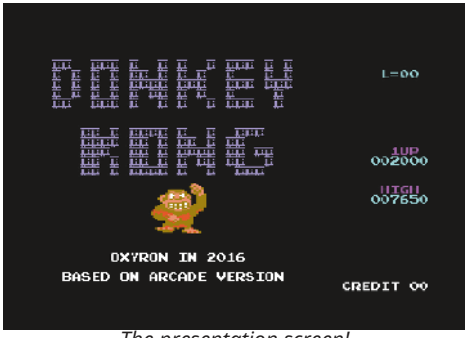
This new version opens with an amazingly detailed introduction screen from talented C64 artist Veto; amid steel girders of an unfinished skyscraper a defiant Kong hurls a barrel toward unseen foes. The accompanying music is C64 musician Linus' perfect arrangement of the title music from Nintendo's *Donkey Kong Country*.

A pre-title graphic screen

At the press of a key we are catapulted into a repeating sequence of screens: high score table with "coins", a 30-second "attract mode" and the title screen.

Applause!
That applause is not just for the existence of those screens, but also for their appearance. The game looks very similar to the arcade version - a point we need to discuss further:

If you remember the original *Donkey Kong* arcade game you'd be correct if you thought its aspect ratio might be difficult to replicate on a Commodore 64. The C64's versatile multicolour mode comes at a cost: the horizontal resolution is halved, so pixels become twice as wide - an effective aspect ratio of 2:1. Images originally designed for a 1:1 ratio appear stretched on the C64, as can be seen in the original Nintendo port.



The presentation screen!

Ocean's 1986 version is closer in appearance to the arcade original, and from a

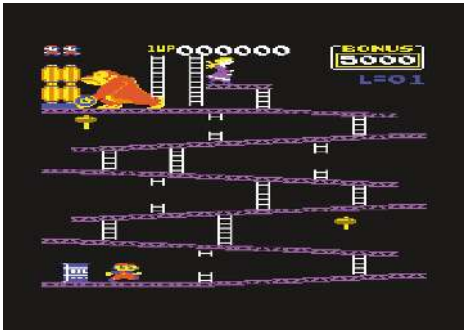
quick glance at Oxyron's offering we could argue theirs is the same except for the positioning of points and credits. However, a closer look reveals that the Oxyron game uses more vertical space than the other two!

The image with the three versions merged clearly shows Oxyron's version extending beyond the top of the visible screen. But how?! The programmer Peiselulli switches off the vertical borders and used sprites to portray the upper section of the building!

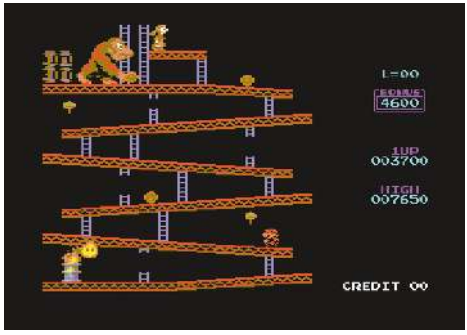
Ok, back to the game. When play begins we are treated with a near-perfect copy of the original arcade game intro sequence in which Kong, cradling your (hopefully unwilling) sweetheart, climbs to the top of the screen then stomps about to partially collapse the structure. As with the original we are then taunted with the message "How high can you get?" before the rescue begins.

The gameplay will be familiar to anyone who has played the arcade original. The animation is fluid as Mario runs, climbs and jumps over barrels. Surprisingly, the graphic definitions and colours are not copied from the original, but reworked in a typical (and in my opinion, more appropriate) C64 style.

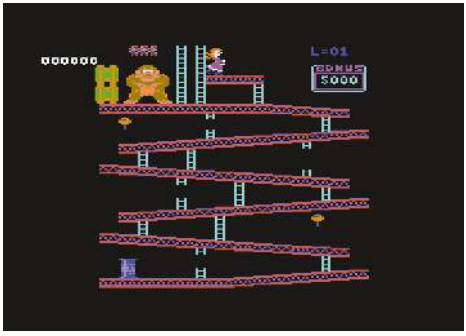
This is what Veto said about the graphics in one of the online forums: *"I was thinking about adopting the original graphics in the beginning but, to be honest, I wasn't really fond of that idea - the arcade graphics are designed for a higher resolution and more [flexibility] with colours. In my opin-*



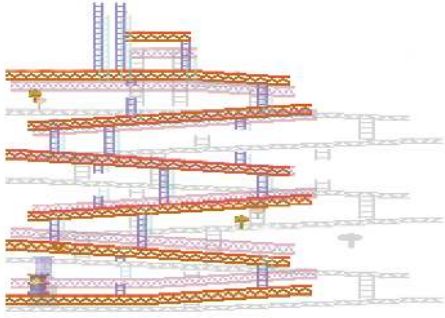
Nintendo version (1983)



Oxyron version (2016)



Ocean version (1986)



Level 1 from the three games, merged

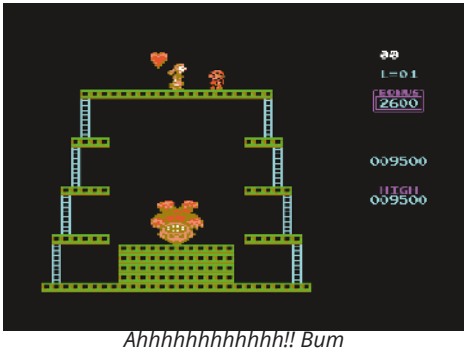
ion it is more interesting to follow a more C64-specific attempt than a graphics downgrade - which is not unusual by the way. My graphics are more inspired by the gameboy version, even though the ape is typical[ly] me, I'd say..."

There is only one thing left to say: this is probably the best arcade conversion to date, and shows how future arcade ports can be done accurately. Well done boys!

Creator's corner: code with Peiselulli

How did you come up with the idea to make a new conversion of Donkey Kong superior to the existing ports?

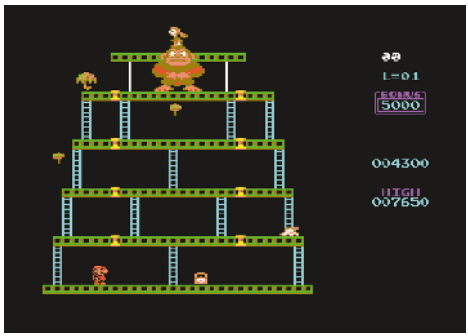
I bought an arcade board called “60in1”. On this board many arcade classics are implemented, one of them being "Donkey Kong". After playing it I felt that this version is much more fun than any version implemented on the C64. As a result, I thought it might be a good idea to make a



Ahhhhhhhhhhhh!! Bum

new C64 conversion that contains the same game logic of the original arcade game. But how would I do it?! Fortunately I found the complete documented source code of the original binary code dump in an online forum that specialised in Donkey Kong.

Up and up!



I can reach you!

What coding techniques did you use to implement such an accurate emulation of the original game engine?

I recognized that it might be necessary to split up the conversion process into two main steps.

First, I would convert all the Z80 stuff into 6502 assembler.
Next, I would exchange the original hardware of the arcade board with the C64 hardware.

But how could I check if the first step is correct and I could start the second one? That was the point at which MAME came into play. I patched MAME so that it continued to emulate the original arcade hardware but the Z80 CPU was replaced with a 6502 CPU. This can be done very

easily because of MAME's modular concept. After that I had my first target platform, plus a decent debugger.

Then I began converting the code step by step from Z80 to 6502, by hand. Because the Z80 has many more registers than the 6502 I decided to replicate those registers in zero page if used regularly. The absolute addresses of the original code were replaced with labels, which allow more flexibility.

Here's some example Z80 code:

```
; else we are drawing a ladder
0DF3 3AB263 LD  A,(#63B2) ;load A with ?
0DF6 D610   SUB #10       ;subtract #10
0DF8 47     LD  B,A       ;copy answer B
0DF9 3AAF63 LD  A,(#63AF) ;load A with ?
0DFC 80     ADD A,B       ;add B
0DFD 32B263 LD  (#63B2),A ;store into ?
0E00 3AAF63 LD  A,(#63AF) ;load A with ?
                           computed above
0E03 C6F0   ADD A,#F0     ;add #F0
0E05 2AAB63 LD  HL,(#63AB);load HL with
            VRAM address to begin drawing
0E08 77     LD  (HL),A    ;draw element
to screen = girder above top of ladder ?
```

When converted to 6502 it becomes:

```
l_0df3:
  lax l_63b2  ; load A with ?
  sbx #$10    ; subtract #10
  stx b
  lax l_63af  ; load A with ?
  clc
  adc b       ; add B
  sta l_63b2  ; store into ?
  txa         ; load A with ?
                computed above (?)
  clc
  adc #$f0    ; add #F0
  ldy l_63ab  ; load HL with VRAM address
                to begin drawing
  sty l
  ldy l_63ab+1
  sty h
  sta (hl),y  ; draw element to screen =
              girder above top of ladder ?
```

This was done for approximately 11,000 lines of Z80 code.

Eventually I had a version that I could play on my patched MAME. The handful of bugs were found very quickly with MAME's debugger.

After this, step 2 started.
The first problem was the screen orientation. On the arcade board the character matrix is oriented vertically, so each successive character address is directly below, not to the right as is with the C64. There are 32 character rows, so the address of a character to the right is obtained by adding 32.

Fortunately I expected this and I had marked this screen address with a "g":

```
  lda #<g_76d4 ; load HL with screen VRAM
                 address ?
  sta l
  lda #>g_76d4
  sta h
```

I had to recalc the hard code address of $76d4 in the original to:

```
g_76d4 = v_base + 20*VIDEO_YS +
         22*VIDEO_XS
```

Because there were many labels like this a script was used to do the recalculation ;-).

This is the way I conditionally assembled either C64 or MAME:

```
!ifdef MAME {
VIDEO_XS=$20  ;step for x dir (to left !)
VIDEO_YS=$01  ;step for y dir (down)
} else {      ;C64
VIDEO_XS=-$01 ;step for x dir (to left !)
VIDEO_YS=$28  ;step for y dir (down)
}
```

However, code that calculated screen addresses during the game had be identified and converted by hand.

The biggest problem of converting to the C64 was its very limited sprite hardware compared with that of the arcade version. This was solved by using a very powerful sprite multiplexor I developed especially for the game.

To achieve a better aspect ratio than that of Ocean's version you had to open the

border for more vertical space. Was your work complicated by this?

Not really. Only that even after opening the borders the screen was still not high enough. The arcade game is 32 characters high and 28 wide. The graphics at the top are limited - which meant I could only replace them with sprites. The graphics such as the scores were moved to the right-hand side. The sprite multiplexor didn't need to handle the sprites' $d010 high bit.

What development environment and tools did you use to code the game? How much time did it take to code the game?

I used *Makefiles*, the *ACME* assembler, the *VICE* emulator and the patched *MAME* for coding under Linux - nothing special. The conversion took about 2 years to complete.

How did you feel about working with Veto (graphic artist) and Linus (musician)?

I'd previously worked on many demos with Veto. The most intensive work we had done together was for the demo "*Coma Light 13*" by Oxyron. I'd also worked on many other games with Linus, such as as my Vectrex conversions "*Fortress of Narzod*", "*Spike*", and "*Minestorm*".

Veto and Linus came in at a very late stage of the project. At first I had used graphics converted from those of the original game. These looked dated and quite monochromatic, but were adequate for use during development. However, at a demo party I asked Veto if he wanted to make some nice graphics for the game - because at this point in time the game was almost playable. He sent me some PNG files and I integrated them with the game step-by-step. He also had some ideas for work-arounds to the C64's hardware limitations; for example, the different ladder colors in each stage.

Since the game and new graphics were almost finished when Linus joined to do the title music I could tell him how much memory he could use for it. The original arcade game lacked a title tune so he came up with the idea to make a title track based on "*Donkey Kong Country*".

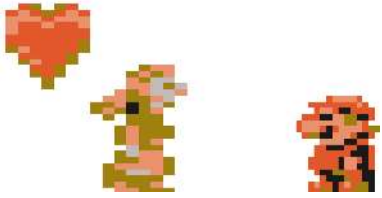
Have you any special information or thoughts about the game that you wish to share with us?

Perhaps a small anecdote: I played the game with a trainer enabled and found a bug on level 22. I fixed it so I could play later levels. But thanks to some Internet research I discovered that the level 22 "kill screen" is a very important part of the game for highscore hunters. So I reverted this change to its original state.

The official version has clicking noises on some hardware configurations. I have fixed them and I think I will release an updated version in the near future.

Have you any other games or arcade conversions planned for the future?

No, I don't think so. I joined "PriorArt", so I think I will continue with some really new developed games in the near future.



The book of Ivan Venturi

Although there were few Italian software houses during the 8-bit era, one that most Italians will remember is Simulmondo. The company's founder Francesco Carlà was widely recognised for his game column that appeared in *MC Microcomputer*, which for two decades was the most important IT magazine in Italy.

Ivan Venturi worked for Simulmondo and was the creator of many C64 games. He recently crowd-funded a book of his memories of that period: *"Vita di videogiochi - Memorie (a 8 bit)"* [Videogame life - Memory (at 8 bit)]. The book was released in limited numbers, and in only the Italian language, so unfortunately few of you have probably read it. However, what follows is a short review of the book which should show you something of the atmosphere of Italy during that time.

In those days it was legal, in Italy at least, to take a commercial game, simply change its title (or even, if you were capable, translate it into Italian) and sell it in newsstand kiosks without giving any royalties to the original authors! Ivan remembers how one particular teacher from the University Of Milan and a Bolognese doctor undertook such activities on an almost industrial scale. Ivan had written a BASIC menu program for the doctor that would be used for one of the "pirate" compilation tapes.

Ivan, a talented illustrator and cartoonist, had the good fortune to learn programming from the computer magazines of that period, and from his older brother who would sometimes borrow his friends'

Cover of the book

computers. Ivan was able to purchase a Commodore 64 after being awarded a scholarship. Soon after he produced a text adventure called *"Fuga da Kreon 3"* ["Escape From Kreon 3"] that was shown to the journalist Francesco Carlà thanks to their mutual friend Stefan Roda.

Francesco told him about his vision for Simulmondo games and Ivan, fired with enthusiasm, began to develop games written mostly in BASIC with some small machine code sections. After a few months he brought his work to Francesco who took one look at them then loaded *Cauldron II* and said, "Unless we can create a game like *that* we won't get anywhere!" This was a traumatic disappointment for young Ivan, but it gave him the impetus to do better. From that point on he impressed Francesco by programming in machine code and draw-

ing well-defined C64 graphics.

He went on to produce the first Simulmondo game that was recognisably Italian: a traditional bowling simulation ("Simulmondo" can be translated to English as "simulate the world").

He was just 17 and this first game netted him 1000€ - a considerable sum of money back then. Another 250€ was earned for a Tombola (Bingo) simulation written in just one month. His next two games were also sports simulations, golf followed by soccer, which was released in time for the 1990 FIFA World Cup, hosted in Italy. It took Ivan one and a half months (instead of the planned five), often working 18 to 20 hours each day to get it ready before Christmas.

Dedicated to me..

Although Francesco had given him a 16-bit Amiga computer Ivan decided to concentrate on pushing the 8-bit C64 to its limits rather than learn an entirely new computer architecture.

F1 Manager was Ivan's next major project. This simulation was five times the size of Italian Soccer but he had a six-month timeframe to complete it before he entered compulsory military service later that year. Then, disaster! The duplicator manufactured an initial batch of 6,000 tapes, but when Simulmondo received them it was discovered that a recording error caused the game to fail to read at datasette counter's 27th position!

By that time Ivan was doing his military service, but thanks to his ability to draw portraits for his comrades he was able to have a private area in the barracks where he used his C64 to work on the graphics for a motocross simulation. When he returned to Simulmondo he gradually moved away from programming into C64 graphics - due to the company having no one else available for that task. After that he held the position of Technical Manager until he left Simulmondo.

This is a book that will draw you into a past world of fond memories and which you will not put down until the final page. Regretfully, it is only those who can read Italian who will be able to enjoy it.

I wish to thank Ivan for giving me one of his personal copies of his book when he learned I'd missed the crowd-funding campaign. For those of you interested in obtaining a copy, he is taking pre-orders for the next reprint.

As this printed volume of ReGame 64 is possible only with your contribution we've decided to give you a PDF version for your personal use. Use the QR-code below to download the PDF to your tablet or other portable device (user **REGAME1**, password **BSJR67DF**).

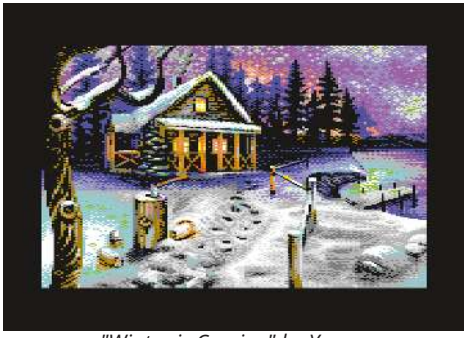
GRAPHIC PIXEL ART

"Song of the dead" by Duce
HIRES + SPRITES - 2016

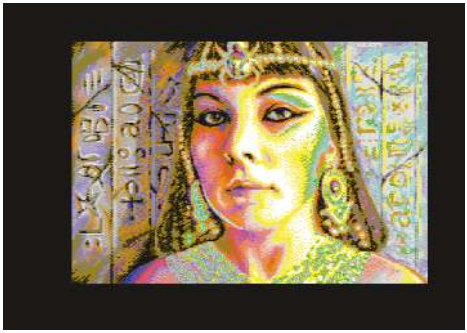
"Them Apples" by ptoing
MCI (Multicolor Interlace) - 2016

"No Shit" by Mermaid & redcrab
Multicolor + SPRITES - 2016

"Snoopy Hires" by Electric
HIRES - 2016



"Winter is Coming" by Yazoo
Multicolor - 2016



"The Scarabeus Queen" by Leon
IFLI (Interlace FLI) - 2016

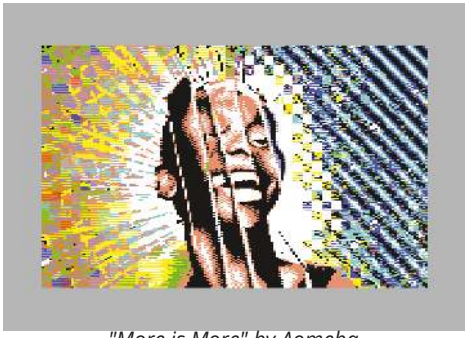
duce: "Truly didn't expect to win a thing with this piece..."
ptoing: "originally I started a new MCOL pic for X, but then stumbled over this pic I started in 2009 I think. It was nowhere close to being finished, but the general feel/vibe was there and I really wanted to see what Asslace looks like on the bigscreen."
Electric: "Not really into NUFLI. I like the crispiness of hires and the limitations of it, Petscii's as well. There's more challenge and they present more 'the core' of C64 to me... of course they don't

eat that much memory either – optimizing is never too oldskool. If I like to do something realistic I rather do it on paper."

Yazoo: "I love to use the built-in modes (have to try hires a bit more sometime - still) - as this is most appealing to me. for me it's the maximum of fun to try and see what i can do in the multicolor mode. if i want a higher resolution or more colors, i'd probably go and do amiga pixeling :-) so usage of nufli is for 99,9% never gonna happen here. I may someday try another mode like afli probably just to have it done at once (or good old FLI) - but most of my pictures will still be multicolor forever i guess :-)"

amoeba: "This picture is an experiment in using the color clashing of the multicolor mode for

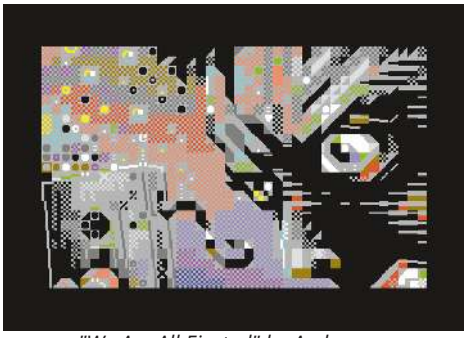
"One Second Demo" by Jok
Multicolor + SPRITES - 2016



"More is More" by Aomeba
Multicolor - 2016

"Snuggery" by Sphinx
Multicolor - 2016

"Concerto" by DeeKay
NUFLI - 2016



"We Are All Ejected" by Archmage
PETSCII - 2016

"BOSSE" by redcrab
PETSCII - 2016

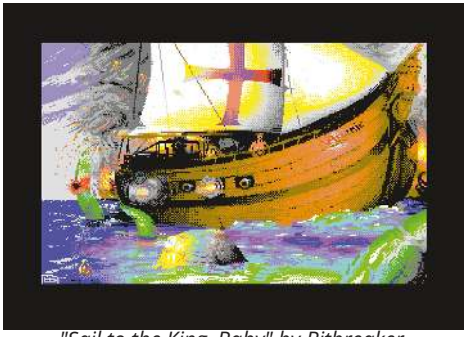
good. The background is done by spraying and filling over and over again in various patterns, causing color clashes and interesting looking crap on the picture. After this, some manua fixing was done."

Archmage: "Dedicated to those who have worked for the facilitation of tape dumping on the C64: SLC, Luigi and Enthusi to name a few."

Duce: "Done back in Feb 2011 and supposed to have some sprites at top border as well."

DeeKay: A NUFLI hires remake of STE'86's Commando Titlepic, Used in Concert (hence the changed logo!), entered into the GFX compo...

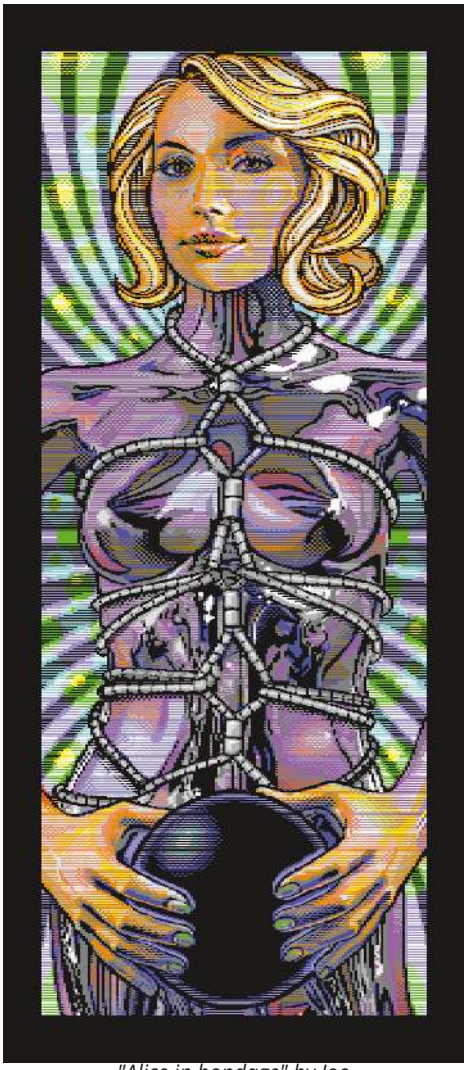
Carrion: "This pic is not finished but I wanted to participate in the compo so much (felt like I promised it to Jazzcat) so here you have the



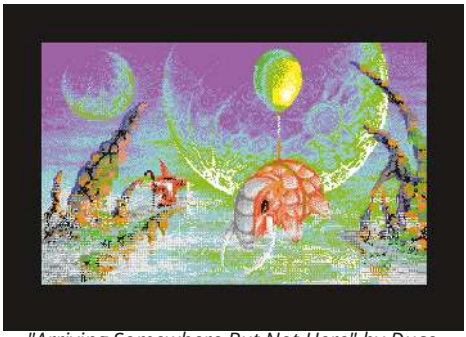
"Sail to the King, Baby" by Bitbreaker
Advanced FLI - 2016

"Sweet-Smorky Dreams" by Leon
IFLI (Interlaced FLI) - 2016

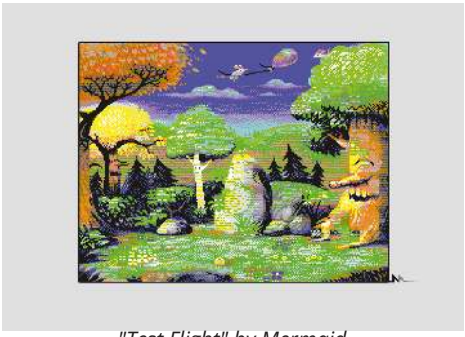
"Searching..." by Leon
MCI (Multicolor Interlace) - 2016



"Alice in bondage" by Joe
Multicolor scroller - 2016



"Arriving Somewhere But Not Here" by Duce
NUFLI - 2016



"Test Flight" by Mermaid
Advanced FLI - 2016

"Take This Life Energy" by Carrion
Multicolor + SPRITES - 2016

"Giantesses" by Animal Bro
Multicolor - 2016

"Breakfast of Champions" by iLKke
PETSCII - 2016

"Coral Reef" by Shine
PETSCII - 2016

result. I started real pixeling 2 days before compo and some ideas and sketching on tuesday. It's the fastest drawn pic by me so far ;) I knew it will be not perfect so as a bonus I decided to go with sprites and top/bottom border (for the first time) and I think I will exploit these areas more. It was so fun anyway to see how this picture grows during the pixeling process that at the end I'm happy I've done it. Even though it looks like a compo filler ;)"

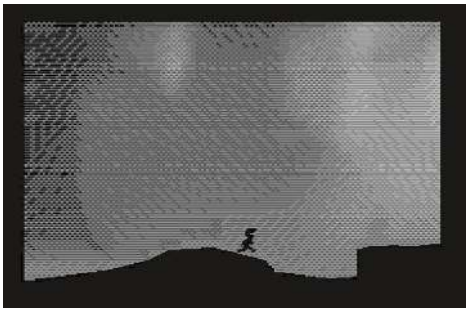
Mermaid [Test Flight]: "Evolved from an MSX-1 picture that I made a few years ago."

Leon [Searching]: "No Copy! (check the concept picture...)" [Sweet-Smorky freams]: "This is based on a photo image. OWN lighting, OWN concept, OWN camera, My photos, OWN items and supplies Own compositions, OWN ideas and My girlfriend!"

LIMBO (?)

Limbo is a puzzle-based platform game by Playdaed released exclusively for the XBox in 2010, and later made available for other systems including PC and PlayStation.
In this game you play a boy who has woken in a dark forest. You set out to find your sister who has become lost somewhere in this dangerous, mysterious and colourless world.

But, why am I telling you about this game? Simple - one of the game's original authors, Søren Trautner Madsen, already well-known in the C64 scene, has received permission to port it to the Commodore 64. He has already produced a proof-of-concept video which can be seen on YouTube! Okay? so listen carefully:



Limbo on C64 (preview)

In the video we see a static screen with a dark color scheme in which the silhouette of a boy runs, jumps and climbs about the landscape just as in the original game. It's a video best described as "Twenty-one seconds of *'WE WANT IT FINISHED!'*"

But can Limbo actually be ported to the Commodore 64? It's a valid question so we will try to analyse the possible scenarios, since, unfortunately, Søren has spoken no further about this project.
From the demo we can see that dark ambient imagery can be created effectively using the C64's limited palette, and animated objects can be rendered accurately with mono-colour sprites. The original game's sound is restricted to ambient effects which maintain its atmosphere of unrelenting horror. The C64's sophisticated SID can easily recreate such effects.

The puzzles are solved using objects, gravity and machines. To make the interactions realistic the game uses a physics engine - which will be a problem for the less powerful C64. It's possible the game could use custom code for each puzzle rather than a generalised engine. A scrolling screen like in the original is also unlikely due to performance issues from combining physics with full-screen scrolling.

The game has many large moving structures and full-screen rotation. Even multiplexed sprites may be insufficient to recreate such objects. Dynamic character-sets could be useful here, but would make development of the game an even more complex undertaking. So the question remains: is Limbo on the C64 even possible?

I think yes. If the atmosphere of the game is recreated then the nature of the puzzles can be reduced to a degree the C64 can manage. Scrolling levels or sections thereof could be made static where necessary. So for now, we'll just have to wait and see...

ADDENDUM

Back in 2003 I began writing a column for SID related topics for **ReLoad** - an Italian paper magazine for the C64. The column used the same title, **SIDin**, as my own PDF magazine that I've been producing since 2002. **ReLoad** had about 42 black-and-white pages and a color cover, probably printed in-house by its owner.

Unfortunately just two issues were produced! I liked the idea behind that magazine so much that in 2007 I began to create my own magazine focused on games and graphics. It has the same A5 format as **ReLoad**, but the pages were designed with colour in mind, and a website was created to accompany it.

My magazine was named **ReGame** because it was intended to show games of the past and encourage people to play games again on the Commodore 64. But showing only old games was not enough, so I decided the magazine would also cover the best images ever pixelled on our favourite 8-bit computer.

So I used **GIMP**, **Inkscape** and **Scribus** to begin making the magazine - but my PC hardware at that time was simply not enough. It took 20 minutes to load the magazine in **Scribus** then another 20 minutes to save the changes (it was like using a Commodore Datasette!). In the end I had to shelve the project until the situation improved.

ReLoad magazine

But during that same period the last issues of **Game Over(view)** were published. This was a C64 disk magazine that focused on new C64 games being created at that time. That idea was a good one, and because of the increasing number of new commercial game releases I realised that approach should be the model for the soon-to-be-reborn **ReGame**.

ReGame collage

Another magazine that helped in this decision was **Commodore Free**, an excellent digital publication still produced to this day and featuring many interviews with people related to the programming and demo scene. But I wanted a magazine with that kind of content on paper...

So, back to the very recent past, there were two items that encouraged the birth of **ReGame**:

Reset 64 Magazine and the fanzine **Freeze64**. I received a specially printed issue of **Reset 64** (issue 8.5) as a stretch goal for the Kickstarter project *"The Story Of Commodore 64 In Pixels"*. For me, the paper version is more valuable than the PDF version. **Reset 64** usually has 20 copies printed, distributed on a first come-first served basis, with everyone else receiving the PDF version. It is a pity not everyone gets a printed copy. Paper is paper...

On the other hand, **Freeze64** is a printed fanzine devoted to gaming cheats (as was common in the old days), new games in production, special interviews, game secrets, game endings and more. It is created by Vinny and you should never miss an issue of this high quality product. It is a pure diamond that you must have on your desk, believe me!

With those last examples in mind, **ReGame 64** is now a reality - but as a book, not a magazine! This may seem odd but it is the only valid way to release it in Italy, even if it is a not-for-profit product! So it will be a book minus an ISBN number (to reduce costs). It will produced as volumes, each with a preface, chapters and bibliography - just like a book. A good one.

As you can see, I want to give a lot of space to the coders, artists and musicians who created the games reviewed in the book. This is by choice, and will be the hallmark of this publication.

ReGame 64 will also showcase the pixelled artwork of the best C64 graphic artists. When I first started planning this publication I intended that they and their work should receive special attention. Additionally, relevant information and public comments from Commodore Scene Database (CSDB) will be reported here.

The front cover of the book is presented in the style of a noticeboard on which are pinned artistic reproductions of the main themes of the reviewed games. This is an evolutionary step up from the original ReGame cover which used vectorised images of the original 8-bit game graphics.

The back cover of each volume will be reserved exclusively for an artist's fantasy composition of one of the games featured inside.

Freeze64 Fanzine

BIBLIOGRAPHY

Caren and the Tangled Tentacles
http://martinwendt.de/caren/
http://facebook.com/carengame (accessible for everyone)

Let's Invade!
http://csdb.dk/release/?id=151230

Donkey Kong 2016
http://csdb.dk/release/?id=151272

Commodore Free
http://www.commodorefree.com/

Game Over(view)
http://csdb.dk/group/?id=4725

Freeze64 Fanzine
http://freeze64.co.uk/

Reset64
http://reset.cbm8bit.com/

Limbo
https://www.youtube.com/watch?v=4FOOeEV36c4

Aomeba	http://csdb.dk/scener/?id=320
Animal Bro	http://csdb.dk/scener/?id=26948
Archmage	http://csdb.dk/scener/?id=14268
Bitbreaker	http://csdb.dk/scener/?id=1678
Carrion	http://csdb.dk/scener/?id=501
DeeKay	http://csdb.dk/scener/?id=8058
Duce	http://csdb.dk/scener/?id=8065
Electric	http://csdb.dk/scener/?id=8064
iLKke	http://csdb.dk/scener/?id=24107
Joe	http://csdb.dk/scener/?id=1672
Jok	http://csdb.dk/scener/?id=5870
Leon	http://csdb.dk/scener/?id=1373
Mermaid	http://csdb.dk/scener/?id=19
ptoing	http://csdb.dk/scener/?id=11032
redcrab	http://csdb.dk/scener/?id=22995
Shine	http://csdb.dk/scener/?id=24193
Sphinx	http://csdb.dk/scener/?id=16048
Yazoo	http://csdb.dk/scener/?id=1127


ALARM