Page 1
PTFC 'Knight Lore 2006'
Page 2
2
Page 3
CONTENTS
3
Page 4

4
Page 5
5
Page 6
1. INTRODUCTION
6
Page 7
1.1 INTRODUCTION
Currently the game go through a moment of glory with the introduction
new tools such as graphics accelerators are 2D / 3D and powerful cards
structuring algorithms.
But whenever the old games, mostly developed during says
80, have a charm which new lack. It has recently emerged
consisting create remakes of old games movement, upgrades
of these games that use of the possibilities offered by the current hardware.
1.2 TARGET
The aim of this PTFC is developing the remake of a classic game of ZX
Spectrum, the Knight Lore.
To achieve as close to the original game results, it was decided by
use a new approach where instead of reconstructing the game again from zero,
we want to maximize the original. Given that this works on
Spectrum missing, we are forced to work on an emulator.
The main idea is to reverse engineer the game for later
diverting the flow of data at the time of generating images, using our
proprietary algorithms to create 3D scene from the data generated by self
game. Thus the gameplay remains intact, only update the
graphics. The main reason for choosing the game Knight Lore is to work with a
pseudo 3D techniques (technique called Filmation), which allow a reconstruction
part of the scene.
Figure 1.1 - Example of objective PTFC
7
Page 8
1.3 TOOLS
The first tool used was the ZX-Spin emulator for processing
reverse engineering. It is a complete emulator at the same time I simple, with which the task
will study how the Knight Lore was made ??intuitively and quickly.
Unfortunately it is a proprietary program and its source code is not available.
There is another great emulator, FUSE, which also could be used for these
necessities (and as a basis for the project), is designed to work under Linux, it is a
excellent emulator, widely used by the community of users
emulators, but currently presents complications in installation and has the
limitation of working exclusively under Linux. The emulator chosen as base
Aspectrum project was the first by having their source and their
compatibility with many existing platforms.
As the MinGW compiler, we chose a popular port tools
GNU programming for the Windows operating system. Along the compiler
Allegro also use the library, which is used by the emulator Aspectrum
Windows. It is a library for programmers C / C ++ development-oriented
video game, freely distributed, and works on the following platforms:

DOS, Unix (Linux, FreeBSD, Irix, Solaris), Windows, QNX, BeOS and MacOS X. It many functions for graphics, sounds, user input (keyboard, mouse and joystick) and timers. It also has fixed-point math functions and floating point 3D functions, functions for handling files, compressed files and graphic interface.
Finally the 3D graphical representation in parallel was possible using the OpenGL library is a standard specification that defines an API multi- multi-platform language for writing applications that produce 3D graphics, originally developed by Silicon Graphics Incorporated. Its name means Open Graphics Library, which translates open graphics library. Among his characteristics we emphasize that is multiplatform (having even a openGL It is mobile), and management of 2D and 3D graphics hardware offering the programmer a simple, stable and compact API. Besides its scalability has allowed development has not stalled, allowing creating extensions, a series of additions of the basic functionalities, in order to take advantage of growing technological developments.
8
Page 9
1.4 TIMELINESS
The first task was to study the game, this was resumed later to some contrasting results. Subsequently the computer ZX Spectrum was studied, especially the assembler code. The next task treatment on engineering Conversely, it was one that required more time, as was the most creative for discover conceptual gaps. Subsequently, a study was conducted emulator and adaptability to our intentions. Finally the task was carried out implementation of the project, which was requiring more time, as was to mostly put to practical much of the knowledge acquired in previous stages.
1.5 STRUCTURE OF THE DOCUMENT
This document is structured along the path, conceptually, which it has led to the completion of the project:
? An introduction to the project.
? A chapter on Z80 microprocessor and computer ZX Spectrum, your history, capabilities and possibilities.
? All the information needed to understand and comprehend emulation and Aspectrum emulator.
? An extensive chapter on reverse engineering and as applied in the operation project to decipher the graphics of the game.
? Implementation of the new environment by listing the steps to get the final result.
? A final chapter discussing the findings, conclusions and future work.
9
Page 10
10
Page 11
2. Z80 AND THE SPECTRUM
eleven
Page 12
2.1 THE Z80
The Zilog Z80 (Z80) is an 8-bit microprocessor was released in the 1976 by ??the company Zilog. Its architecture is characterized by being halfway between the organization of storage and general purpose registers. It was popularized in the 80s through personal computers like the Sinclair ZX-Spectrum, Amstrad CPC and MSX computers. It is one of more processors market success, of which there have been many clonal versions, and continues It is used extensively today in a variety of devices wardrobes.
Figure 2.1 - Z80 Pickup
The Z80 was designed mainly by Federico Faggin, who was working on Intel as chief designer of the Intel 4004 and Intel 8080. In 1974 left the company to found Zilog and began work on the design of Z80 based on experience with the Intel 8080.
The Z80 was designed to be compatible at the code level with the Intel 8080, of

so that the majority of programs for the 8080 could function on it,
especially the CP / M operating system.
The Z80 had eight fundamental improvements over the Intel 8080:
? An enhanced instruction set, including new records index
IX and IY and the instructions needed to handle them.
? Two banks of records that could be changed quickly to accelerate
responding to interrupts.
? block move instructions, E / S block and byte search.
? bit manipulation instructions.
? An address counter for DRAM refresh integrated in the
8080 had to be provided by the support circuitry.
? single 5 volt supply.
? less need auxiliary circuits, both for generating the signal
Watch as for liaison with the memory and I / S.
? Cheaper than Intel 8080.
12
Page 13
The Z80 eliminated Intel 8080 quickly to market and became one of the
Popular processors 8 bits. Early versions operating at 2.5 MHz,
but its speed has increased up to 20 MHz. Thus, most commonly used version,
Z80A running at 4 MHz.
In the early 80s the Z80 or versions of the same clone was used in
many home computers. Later, in the 90's, the Z80 was used in
consoles Master System, Game Gear and Sega Mega Drive. Game consoles
Boy and Game Boy Color of Nintendo use a variant of the Z80 made ??by Sharp.
Currently part of the range of programmable graphing calculators Texas
Instruments used a cloned version of the Z80 processor as manufactured by NEC
principal. Furthermore the Z80 microprocessor is also a popular for use in
embedded systems, field where it is used extensively.
Despite being an 8-bit microprocessor, the Z80 can handle instructions
16 bits and can address up to 64 Kb of RAM. One of the characteristics more
noteworthy is that it has instructions Intel 8080 as a subset, so that
some Z80-based computers could run programs designed for the
CP / M. This has made ??the Z80 instruction formats are quite complex,
as they have to maintain compatibility with 8080. However, the Z80 has
managed to improve the speed Intel microprocessor, it has added new modes
addressing and contains a set of broader instructions.
The Z80 registers structure is composed of a main bank, other
alternative and finally bank comprises special registers. The existence of
Reciprocating Rig speed improves the presence of interrupts since
It lets you change from the main to the alternate bank.
Figure 2.2 - Records Z80
13
Page 14
The records of the main bank are general and 8 bits. They may be taken by
couples, being then IX and IY registers indices. A serving of registration
accumulator. The R stores the memory block whose refreshment is going to proceed. He
SP is the top of stack pointer. The PC is the program counter. F contains the
also called flags or status bits.
Figure 2.3 - Block diagram of the Z80
14
Page 15
Sinclair ZX SPECTRUM 2.2
The Sinclair ZX Spectrum was a 8-bit microcomputer based
Zilog Z80 microprocessor manufactured by the British company Sinclair Research and
launched to the European market in 1982. The hardware was designed by Richard
Altwasser and software by Steve Vickers. The Sinclair ZX Spectrum was the
most popular domestic microcomputer 80s.
Figure 2.4 - The Sinclair ZX Spectrum 48K
Its main features were:
? Microprocessor Zilog Z80 to 3.5 MHz (bus and 8-bit data bus
16-bit addresses). Also called a chip containing ULA
(Uncommitted Lodge Array) reducing the number of chips required.

? Maximum resolution graphics (and only) of 256x192 pixels. He had an ingenious how to implement the video with 16 colors using only 8 KB of memory RAM. The video was modulated signal for display on a TV ordinary.
? Various configurations with 16K or 48K RAM.
? 16 KB of ROM (including a BASIC language interpreter SINCLAIR developed by the company Nine Tiles Ltd. for Sinclair and it was a evolution already developed for previous two commercial machines the brand, the ZX-80 and ZX-81, and the Spectrum's successor).
? rubber keyboard built into the computer (model 16K and the first version of 48k. A model incorporating an enhanced keyboard led the name of ZX Spectrum Plus).
? storage system common cassette tape audio at 1200 baud (The rate supported by the operating system in ROM, but had games used their own system load "turbo" faster, although more prone to errors load).
fifteen
Page 16
All these features became the ZX Spectrum in a very affordable equipment, microcomputers that brought a large number of people. Over the years were appearing various peripherals such as Interface 1, floppies (Discovery, Microdrive), light pens, joysticks and printers.
The Interface 1 was a RS232 port gave the Spectrum network features, and allowing interconnect up to 64 of them.
The Microdrive was allowed to read a unit of magnetic tape cartuchitos endless. Each cartridge stored 85 Kb. Its cost was much lower than that of a floppy drive of any other computer of the time. The cartridges were based on a ingenious and simple idea once: a small magnetic tape wound endless, which is He is traveling at high speed by the action of roller incorporating units reading, being read or recorded information by a head. Cartridges microdrives tape contained about 200 inches, which was shifted to 28 inches per second. In less than 8 seconds, the tape had come full circle, which which meant carrying around 15 Kb. per second. However, the high Speed ??attached to friction with the read / write head, caused wear very fast tape did not read well and was cut quite often.
Figure 2.5 - ZX Spectrum with Interface 1 and Microdrive
The Spectrum, despite its obvious limitations, was a boom. It was sold in more than 30 countries and became the leader in Europe outselling the Commodore 64, which was technically superior.
16
Page 17
17
Page 18
3. EMULATION
18
Page 19
3.1 This is an emulator
An emulator is a program designed to recreate internally operation a different architecture to that in running. The emulator is simply a program without hardware parts, using the resources of the machine where runs, simulates the behavior of the CPU, memory and other elements of a given machine.
For example, a ZXSpin Spectrum emulator as it is a program designed to emulate a Z80 micro (the Micro Spectrum), a ULA (the "chip graphic "Spectrum), a tape drive or disk, etc ... Just as There ZXSpin, NESticle is a NES emulator that emulates the NES Micro, the NES chip is capable of running games NES cartridge, etc.
Normally emulators are programs that simulate a single architecture, although in some cases there are "macroemuladores" capable of emulating multiple systems,
such as MAME (Multi Arcade Machine Emulator) that emulates variety different microprocessors, which allows many machines recreate recreational. Note a very important detail: an emulator is simply a program

as any installed on the machine. The difference is that instead of editing is
text, make calculations, draw, or play (like other programs), which makes it
behave as would the emulated system.
There are currently many emulated systems:
? Consoles: NES, Superna, Game Boy, Nintendo 64, Master System,
Megadrive, Game Gear, Saturn, Atari 2600, Atari 7800, Lynx, Neo Geo,
TurboGrafx ...
? Gambling machines: Emulators MAME and Raine.
? Computer: Spectrum, Amstrad, Commodore, MSX, Atari ST, Amiga, PC, etc.
3.2 How a EMULATOR
An emulator, roughly simulates a microprocessor. For example, it is possible
make a program that reads instructions Z80 microprocessor, the understanding, the
run, and save the results on executions.
Z80 emulator can be implemented in various languages ??like C, Visual
BASIC, PASCAL or assembly (to name a few common languages) who understands
machine code instructions provided Z80 emulator, the run and
change the emulated microprocessor records, leaving all records
As would be emulated in real Z80 if you executed the same code. Thus,
You can make a program of Z80 machine code, and run it on an emulator,
19
Page 20
or a real Z80, the same results were obtained in exactly the different
microprocessor registers. Recreate the hardware emulator system.
But it's not enough to just recreate the hardware of a machine (and only
emularíamos a microprocessor or a graphics chip, and the latter do not themselves
they are able to do anything), so that the emulator really can do something, you need
a (more or less complex) Operating System. This system used to be stored in a
chip ROM (read only memory) with the basic functions engraved on it.
Formerly, most notably in the case of spectrum, the system concepts
Basic operating and interpreter were mixed so that it becomes difficult to make a
distinction.
When starting a Spectrum, for example, which is a ROM is 16KB (16384
bytes) containing the start of the machine, the BASIC interpreter and all
functions necessary to load and run software Spectrum. This ROM
reality is a memory chip with Spectrum Operating System (program
written in machine code Z80) engraved on it. This program
has practical differences with a game or a program on tape or disk Spectrum,
its function is simply not play, but provide us with a management interface
Spectrum (BASIC) and instead of being recorded on a tape, which is in a
read only memory. That is, there was a person (or more) you set (as if
was a game) the start of the machine, the BASIC, etc., menu and put it together with
a 'assembler "for the code stored in the machine
ROM.
The emulator replicates a micro Z80 (and the other components of Spectrum), and
first thing is loaded from the file contents of the ROM Spectrum
(Rom48k.rom) so that the emulator, like real Spectrum, bootable,
executing the instructions of the Spectrum ROM through its "microprocessor
emulated ". For this file with a chip ROM Spectrum is caught, is
inserted into a reader / writer of memoirs, and read, recording the content of 16384
bytes in a file. Each byte of the file you could say that is an instruction
ROM program Spectrum.
Figure 3.1 - Basic diagram of a computer Spectrum
20
Page 21
The central part is a micro Z80 which sees the ROM and RAM continuously
as its entire memory. That is, seen 64KB memory of which
first 16K are the content of ROM chip, and the following chip 48K
RAM. When you turn the Spectrum it initializes and displays the BASIC because it
starts
executing instructions from the memory address 0, which is the principle of
16K ROM, that is, the BASIC interpreter.
That's why, when we turn on the computer, "run" the ROM. When you turn
a Spectrum (who lost his off all power) all records

Z80 microprocessor worth 0, including one called PC (Program Counter or Program Counter) that is pointing to the next instruction to the Z80 must read and execute. A microprocessor operates broadly as follows:
? Read instruction pointed to by the PC registry.
? Increase PC to point to the next instruction.
? Run the newly read instruction.
? continuously Repeat the previous 3 steps.
So, when you turn on the computer, PC is 0. Being mapped in the ROM memory location 0 (through wiring hardware memory chips in the Spectrum plate), what happens when the computer is that counter program (PC) is pointing to the beginning of the ROM, which is why it runs the ROM step by step instruction by instruction, every time you turn it on. For him Spectrum all memory chips inside are like a big trunk 64KB continued, which is achieved by wiring the different chips right pin of the microprocessor. Broadly, the pin data and microprocessor addresses are connected to different memory chips so that when the micro reads data from memory, he sees everything as if it were a single 64KB memory chip. This is achieved with a simple design process (to do Computer scheme before it's built) known as "memory mapping".
In the Spectrum memory map, the first 16KB are the ROM (which is in a chip apart, but as we have seen is something that the Spectrum does not distinguish, and that displayed as a continuous memory section from position 0 to 16383 of his "full trunk" 64KB) and then comes the RAM, from the 16384 position. That's where programs are stored, the screen graphics (in a piece determined that memory), etc. This RAM is where the BASIC interpreter introduces programs for execution.
These programs can come from various input / output (Managed by the Z80) as the keyboard, disk or tape, etc.
It should be a special mention of that part of the RAM (from the byte 16384 to 23296) is connected with the ULA, the "graphic" Spectrum, and chip responsible for converting the content of this "videoram" or VRAM video signals for television. When the games draw graphics, sprites or anything else screen, they are actually writing bytes of memory in these positions, the ULA shown on TV in the next screen refresh.
21
Page 22
Seen in a simple way: to enter a numeric value (eg 1) some direction on this side of the RAM, a point immediately appears in our monitor, since the ULA is continuously "scanned" the videoram (of independently of Z80) to reflect on the monitor or television all values Numerical we put into it. In the PCs it is the same; to write a value number one position in memory, points appear on the screen. According to We write what direction appear in one place or another screen. In some video mode (320x200, for example), write position
A000h
brings up a
color dot at position (0,0) of the monitor, do it in
A001h
, It does appear in
(1.0), and thus a pixel after another.
This gives us a software that emulates a micro (Z80) and thanks to the ROM the same start and shows and lets you use the Spectrum itself as Spectrum could use a newly started. The emulator making, in short, is read keyboard events and communicate the micro emulated in the same way you It made ??real by keypress Spectrum. At the same time, reads memory Spectrum screen content, and displays it on your monitor as it "Read" TV. The virtual microprocessor, meanwhile, engages simple and only execute instructions, either ROM or a game we load or execute him.
The ROM is very important because it indicates how it should behave the microprocessor at all times, how to serve the interrupts received by

keyboard and controls, etc. For example, a computer ZX Spectrum and Amstrad CPC have the same microprocessor, Z80, yet are substantially different. Why? Because apart from the accompanying micro circuits are different, the ROM is completely different, so that changes the System Operating, memory addresses where the screen data is stored, etc.
The mere fact of being able to use a particular architecture and operating system You may already be a great incentive, but if this were not enough, in most cases also we can use all the original physical system software in the emulator.
Whether a cartridge, a tape or disc, the objective is to obtain a copy format digital (in a file) of data, so that they can be loaded in the emulator.
Consider the different types of software to emulate:
ROMs of the system: as we have seen, the machine ROMs are obtained spilling the contents of the memory chips of the same, where they are stored programs into machine code by readers of memoirs or the like.
Figure 3.2 - circuitry of a ZX Spectrum
22
Page 23
ROM cartridges: console cartridges are typically not more that supports plastic in a certain way, that inside contain memory chips (Same as the ROM chips) electrically connected to the metal contacts or pin protruding plastic cartridge. Actually console games are not but the system ROMs as consoles really do not usually have ROM (if ignite without play inside, do nothing), and we who introduce a ROM (which is actually the game) to enter the cartridge. That ROM instead of an operating system, is a game. Recall that the microprocessor all he can do is execute instructions in machine code, it's a game, or an BASIC interpreter. In some systems do have prerecorded ROM in the machine able to perform tasks when you do not enter game. In this manner, clonic NES consoles could carry hundreds of games recorded on a ROM chip internal, so that when the turn on without introducing a cartridge, the chip is active and content is executed. When you insert a game, instead, the chip actually becomes the memory of the machine (and therefore runs) is the located inside the cartridge. To dump games to disk files so that can be used in the emulators, simply remove the cartridge, remove the chip memory with the code recorded game, and as in the case of ROMs system, overturn a file with a memory chip reader. Another option would be to use some readers of cartridges that are inserted into the machine and allowed to burn the contents of the floppy ROMs. These devices were mainly used as "copycats". In general, in the world of the emulation ROM is called Full dump file from a memory disk. ROM files have different formats depending on the architecture of the machine. For example, the file extension smc
will be a ROM cartridge Superna, SMD extension Sega MegaDrive, the nes extension a game of NES, Gameboy gb extension, etc. Likewise, there You roms at the Spectrum, obtained from cartridges Interface 2 (a peripheral Spectrum that allowed load cartridge games, doing exactly the same function insertion of the cartridge in a console). The arcades are plates with chips and circuits (in the style of a console) where Games also usually recorded on ROM chips, or are introduced as if cartridges were. The procedure to remove the ROMs is similar recreational the cartridges in most cases.
Figure 3.3 - Circuit NES cartridge
2. 3
Page 24
Tapes: The tapes contain the same software Spectrum cartridges console, only instead of being recorded program on a chip, it is recorded magnetically on tape. The tape contains recorded audio has a speed determined and with a particular format so that the computer can read (and record software also tape). The reader reads this computer tapes and audio interprets the instructions stored in memory. Spectrum tapes often be files with extension tap or tzx, which were obtained using a program Audio recording (with which are saved to disk in * .wav or * .voc). A Taper program as MakeTZX later or in charge of analyzing this file

wav / voc and make it a TAP or TZX file data with the program in a
perfect and readable format Spectrum emulators. A file containing TZX
as the tape have physically recorded, but in the form of ones and zeros in
Instead of audio (so a TZX may occupy about 100KB while the WAV deals
up to 10MB, and are equivalent). The most interesting thing is that from a file TZX
we can get the tape WAV (to re-record) or may "play"
TZX files with programs like Playtzx, and load the game in a real Spectrum
with an appropriate cable.
This file format (TZX) is the most important in the Spectrum. This is so
because as exact copies of the tapes, and from which it can rewrite a
original tape, going to TZX format available all games, actually
we are preserving and avoiding the loss and degradation of the original tapes with
time. If a tape, over the years, loses its magnetic properties and the game does not
freight, we can re-record on the original tape using a cassette recorder and
TZX tape file obtained from the Internet.
It notes that the TAP and TZX files are not ROMS. They are "tape file"
or "tape files". The term ROM, as we have seen, is associated with programs or games
obtained from hardware, and does not apply to these files.
Discs: As the tapes, disks of different systems (including 3
inches) can be passed to disk files with the appropriate disk drives
programs that interpret and record in the appropriate formats. For example,
.dsk, .tcl and .trd files are disk images and file passed Spectrum
used by emulators. As is the case with ribbons from a DSK
we can recreate a disk formatted Spectrum.
Figure 3.4 - Disk drive a Spectrum +3
24
Page 25
Snapshots: they are memory dumps Spectrum already loaded with games
from tape or disk. That is, suppose that in a Spectrum with 48KB of memory
(ZX Spectrum 48K) loaded from tape a particular game. This game (which,
certainly occupy less than 48KB) is stored in RAM ready for Spectrum
played (for example, in the game menu). If at this time, store the
RAM in a file, and simultaneously store the complete state of the
CPU, we have a "copy" of the machine status. When you load this file in another
the same machine or on an emulator, will the target system in the same state
accurate than was the original machine. That is, before the menu, or play in the
point where we recorded.
Snapshots (which are not ROMs) are usually obtained in carrying tapes
emulators and recording the contents of memory to disk in a special format
by type of Snapshot. We .Z80 files (which began using the emulator
Z80 for MS-DOS), files with extension and sna sp (used by ancient emulators)
szx and files with extension, among others. This file format is not ideal for
preserve games, they do not allow recreate the tapes, only play a way
quickly and directly to the games. In general we can save and load snapshots in
emulators a very fast (as files) and at any time,
so that they can be used as a method to "record the items" and continue
later.
Figure 3.5 - Diagram of the types of software to emulate
25
Page 26
EMULATOR 3.3 ASPECTRUM
Aspectrum is an emulator developed by Alvaro Alea and Santiago Romero,
architecture and compatible with any operating system, specially designed for
to be able to survive over time. Currently, although it has not completed its
development, is functional and has been tested with over 450 games. It is also an
emulator
GPL: you can copy, modify, add new features without
to answer to anyone.
Figure 3.6 - Linux Aspectrum
Figure 3.7 under Mac OS Aspectrum
It is written entirely in C language (without a single line of assembler), you can
compiled perfectly on any existing platform that supports any of the
graphics libraries that you can use: Windows, Linux, BeOS, various consoles, etc. This

It is important because there are platforms that have Spectrum emulator (or at least it is GPL) and by Aspectrum this problem can be remedied.
Aspectrum requires one of the two most popular graphics libraries for implement its graphics, SDL or Allegro. While it is true that not all platforms support these libraries since the graphical code represents only 2% of the code emulator (which is written in C), it would be easy to port to phones, PDAs or other devices, rewriting only the graphics. It has now been compiled emulator MS Dos, Windows, Linux and Mac OS smoothly and performance optimum.
Failure to rely on a single architecture gives an advantage over many emulators that base part of their code written in assembler.
3.3.1 INTERNAL OPERATIONS
The inner workings of the emulator, as we were interested, was the area where the main iteration is performed. This consists of a loop that is repeated until ending its execution. In Figure 3.8 schematically shown its operation.
26
Page 27
Figure 3-8 - main loop of the emulator
The function
emuMainLoop ()
It represents all the sentences that make possible a iteration of the emulator.
3.4 COMPILATION
Once the change is only necessary to compile. To do modify the Makefile that brings the emulator, you must add the necessary links to the libraries OpenGL, GLU and GLUT.
Figure 3.9 - Contents of the Makefile
Changes made to the Makefile, see Figure 3.9, consisting of add section
LFLAGS
the arguments required by the option
-l
, for
specify the necessary steps to successfully compile the code libraries. He argument to include the OpenGL library, which is what allows you to draw in 3D, is
-lopengl32
.
By arguments
-lglu32
Y
-lglut32
We specify that use the product
GLU and GLUT libraries respectively. These libraries allow us to add new features drawing, event handling and defining windows.
27
Page 28
28
Page 29
4. REVERSE ENGINEERING
29
Page 30
4.1 basics of the game
In the game we control a character who must cross different locations looking for a series of objects and place them in a room in particular the aiming to break a curse that transforms him into a werewolf.
The stage is divided into more than 200 rooms which consist of backgrounds and objects.
The backgrounds and objects are represented in the program code in the form of sprites, with its size and graphic properties. The funds are usually the walls and doors. The objects are numerous: blocks, fire, spikes, guards, etc. As Room example can see in Figure 4.1 that contains as the funds walls and two doors, and as a group of objects arranged in a grid block

3x4 regularly.
Figure 4.1 - Game screen with its elements identified
30
Page 31
4.2 - TECHNICAL Filmation
The Filmation technique is a special system management and graphics
location at which the game is played, which is displayed on a dunk
isometric perspective and using sprites technique. This technique produces a vision
three-dimensional environment. Do not confuse the Filmation technique (whose name
It comes from the similarity to "shoot" a film) with the management of sprites. A
defect
Filmation system is the need to operate with it in a two-color system, is
say, a color ink and paper of another. This is so because the doll or
Character sprite we handle does not move horizontally or vertically by
screen but because it is an isometric projection, will move
vectors 30 ° inclined to the horizontal screen. This makes it virtually impossible
in our case, because color management organization separate from the
memory attributes to display on the Spectrum.
The basic concepts used in the art are:
? Graphic: The drawing of a doll, car, ball ... located in memory
computer and we intend to move or paint on the screen.
? Sprite: Graphic defined in memory in a special way and managed
Also a special routine, you can move around the screen without impairing the
that it had. Also, when moving, it creates the illusion that the
sprite is at an earlier level to what we have on the screen. Let's say
passes "above" it. Do not confuse graph sprite as
are two different things.
? Shadow or sprite mask: The sprite is stored in a memory
especially, on the one hand as if an ordinary graph it were and other
Also defined is the "shadow" or mask needed to manage
sprite. This information is encoded mask opacity or
Transparency of certain areas of the graphic.
Figure 4.2 - Elements of Filmation technique
As shown in Figure 4.2, the graph corresponding to a magnifying glass is
comprising its sprite and mask. The mask to define which part of the sprite
We want it to be dull and we want to be transparent. At this level of code
It takes place, as we can see in Figure 4.3, making bit operations between
sprite mask and then apply them to the screen.
31
Page 32
Figure 4.3 - Operations Filmation
The result of these operations can place the sprite on the screen without impairing
the rest. For that:
? Make the 1's complement of the mask, reenplanzando 1 to 0 and vice versa.
? To the above result it is applied to the logical AND with the portion of the
screen where you want to work.
? The latter result OPERATION OR logic receives the graphic for the
sprite end.
Notably, as shown in Figure 4.2, the size of the mask is
slightly higher than the sprite. The reason is to get to perform its intersection with
the
sprite with a black silhouette them easier to see and also embellishes.
4.3 REVERSE ENGINEERING
The purpose of reverse engineering is to obtain technical information from a
product accessible to the public, in order to determine what is fact, what makes
work and how it was manufactured. The most common products that are subject to the
reverse engineering are computer programs and electronic components.
This method is called reverse engineering because it moves in the opposite direction
routine engineering tasks, which consist of technical data used to develop
a product. In general if the product or other material that underwent
Reverse engineering was obtained properly, then the process is legitimate and
legal.
Reverse engineering is a method of resolution. Reverse engineer something

It involves further study of its operation, to the point that we can
come to understand, modify, and improve this mode of operation.
In particular in this project we used the following techniques engineering
Reverse:
32
Page 33
? study prior documentation, especially documentation on
Static data structures of the game. See section 4.4.
? Study program source code to understand the functioning of the
routines that were responsible for the graphic management. See section 4.5
? Modification of program data in real time to see the changes
arise, and deduce why the induced effects. See section 4.8.
4.4 REFERENCE INFORMATION
The reverse engineering process started from the binary disassembled
game and two documents filmation On Neil Walker and Knight Lore data format of
Jon Christopher Wild. These writings contain the basic information as required
play the static information in memory.
Nothing more start the game, we have in memory all the static data
Room sorted by location, so compacted, representing
initial state of each room. The information that represents is:
? location identifier
? Move to the following address
? Size and color
? Funds
? Objects
An example of a room as is stored in memory would be as follows
(bedroom
00Eh
):
Figure 4.4 - Static data room 0Eh
The first line contains the identification, color and size of the room and
a reference to the next room while the second contains information
on funds and objects.
? The first byte corresponds to the identifier, which is unique for the whole room,
in this case is the
00Eh
.
? The next byte corresponds to the displacement to be applied to the pointer
program to place in the next room, which in this example is
00Bh
(11 bytes).
? is presented a case in which we compact various data in the third byte in
one byte. Specific
015h
It represents
00010101b
which, applying the
pattern in Figure 4.5, we get the information about the color
dimensions.
33
Page 34
Figure 4.5 - Standard room attributes
Color is
101b
corresponding to the first three bits, the size will be
10b
That
obtained from the following two bits (different room types are
presented in Figure 4.6) and the remaining bits may be ignored since no
They mean nothing. As we can see, in a single byte we can store much
information and, as we shall see, was something quite common because of the
technical limitations of the time. On a practical level, its handling is achieved
easily using bit operations.

Figure 4.6 - The different types of rooms
Once treated the above data is passed to read data from the backgrounds and objects, bytes that are remaining until the start of the next location. Firstly listed funds, where each byte represents the identifier of a fund until appears one value
FFh.
The byte with that value is ignored and simply determines that we have the funds and starting the section of objects.
In the example of Figure 4.4, we note that the following bytes to the bookmark Startup objects are:
001h, 003h, 00Dh, 0FFh
These identifiers represent the arc this (
001h
), The arc west (
003h
) And a corridor
(
00Dh
) Bottom, as shown in Figure 4.7:
Figure 4.7 - Funds room 0Eh
Reached the point of reading objects, the process returns to require the byte division. The method is to get the first byte, broken, which we indicate the object identifier and quantity (a location may contain several objects of the same type). The number n of a descriptor indicating that the next n + 1
3. 4
Page 35
Descriptor bytes contain the coordinates of such objects containing that particular room. As can be seen the maximum number of replicas an object is 8, then 3 bits are arranged to indicate the number. That does not stop placing
more than 8 objects in a room can be achieved by adding as many descriptors objects as needed to reach the required total number of objects. The pattern It is specified in Figure 4.8.
Figure 4.8 - Pattern objects
In the example we used so far (Figure 4.4) we see that the bytes after the end-of funds are:
053h, 012H, 01Dh, 02Ch, 023h
Byte
053h
represents the object descriptor, broken in bits obtain the value
01010011b
. The last five bits represent the object identifier, in this case It will be the object identifier
01010b = 0Ah
which is fire (see Figure 4.16). The
first three bits contain the value
011b,
corresponding to the number of repetitions
object. That will mean we will have four objects (n = 3, 3 + 1 = 4), which shall Fire type.
The next four bytes contain the coordinates of the object of fires type:
012H
=
00010010b
->
(X, y, z) = (2,2,0)
01Dh
=
00011101b
->
(X, y, z) = (5,3,0)
02Ch
=

00101100b
->
(X, y, z) = (4,5,0)
023h
=
00100011b
->
(X, y, z) = (3,4,0)
In this example there is no longer objects, but it could happen that do exist. In that
If we would still read bytes with the previously established pattern, descriptor byte and bytes
specifying consecutive coordinates. At this point we already have
all static information representing a room, specifically the look
graphic example is as shown in Figure 4.9
35
Page 36
Figure 4.9 - Room 00Eh
But this is not enough to know the status at all times in a
location, because we have obtained the initial data, which are static. The elements
the fund will not change position, but many other objects (can move, the
the main character can take, there are blocks that disappear when placed above the
main character, etc). Figure 4.10 shows the same room but a view
Moments later, with a different arrangement of objects.
Figure 4.10 - Double 00Eh
4.5 ROOMS IN MEMORY OF WORK
With information regarding the locations that the game is not initialized
enough room to graphically capture, as these data do not contain the
changes experienced by objects with the passage of time or interactions with other
objects. Just refer to the initial situation. Furthermore, we have binary
disassembly of the game and with certain specified functions and memory maps.
One of these areas of memory was working memory (mem scratch) and that was
36
Page 37
We deduced that dynamically reflect data locations. A
as simple check and confirm it changes arising during the
During a game clarified our assumptions. We do this through the
ZXSPin emulator debug function. We knew where the changes occurred in the
working memory (which is located between the memory addresses
5BA0h
Y
6107h
)
but we did not know as. After many hours of study, and watching it
information about the source of disassembling game, we called attention
a feature called RetrieveScreen . So, we decided to examine it to find out if
was that data stored in each memory location, and how he did it.
It translated into pseudocode looks like this:
Place the pointer working in the id of the first room;
Repeat {
Id jump to the next room;
} Until the pointer pointing to the current work room
Get the attributes of the room
Storing the attributes of the location;
The above snippet runs the initial static data room in search
of the room where the main character enters. Once you get localized
the attributes of the room (see Figure 4.5).
The following excerpt is pseudocode:
For each fund do {
Get your sprite and information about it;
Get its dimensions;
Get his position;
Calculate HD 3D position;
Storing the information obtained;

}
For each object to {
Get your sprite and information about it;
Get its dimensions;
Get his position;
Calculate HD 3D position;
Storing the information obtained;
}
The above two fragments, are responsible for acquiring, for each fund and then by each object, all properties and attributes, and then store the information obtained.
37
Page 38
4.5.1 TREATMENT OF FUNDS
RetriveScreen , for each fund through its value, see the table of funds, alamacenada statically from the direction
6CE2h
(See Figure 4.11).
Figure 4.11 - Table pointer funds
This table allows the address data of a fund just
adding a base address value of each fund. For example, to know the
address where to find the data bow this fund (see Figure 4.7), sufficient
adding to the base address of the table of figure 4.11, which is
6CE2h
The identifier
this arc multiplied by
02h
(For each direction requires two bytes for
store) which is
001h
The result will be
6CE4h
. In the latter direction
We find the data that define the arc. Data defining funds
stored in the format shown in Figure 4.12.
38
Page 39
Figure 4.12 - Data format of a sprite fund
The result will be one or more groups of 8 bytes, each group follows the format
Figure 4.12. This format defines:
Byte ?
00
id contains the sprite.
? Bytes
01
,
02
Y
03
containing the X, Y and Z of the sprite.
? Bytes
04
,
05
Y
06
define the dimensions of the sprite.
Byte ?
07
contains a number of flags: the vyh indicate if active one
horizontal or vertical rotation of the sprite, the rest are bits that do not interest
us
as they are for internal use of the game.

Finally the example data, the arc this (Figure 4.7) are:
002h
, 0C4h, 073h, 080h, 005h, 003h, 028h, 010H,
003h
, 0C4h, 08DH, 080h, 005h, 003h, 028h, 010H,
00h
Observed in the previous line that this arc is formed by two sprites, the
002h
and he
003h
, Which could be seen in Figure 4.13.
Figure 4.13 - Sprites forming the arc this sprite
39
Page 40
The end result in the working memory will be shown in Figure 4.14.
Figure 4.14 - Data bow this object in working memory
We can see that there are more bytes in addition to the boxed in Figure 4.14, are
data used by the game to other aspects of objects that we do not we
interest for 3D visualization.
4.5.2 - TREATMENT OF OBJECTS
RetriveScreen , for each object through its value query object table
presented in Figure 4.15.
Figure 4.15 - Table pointer objects
This table allows the address data of an object simply
adding to its base address value of each object.
40
Page 41
Figure 4.16 - Fire Sprite object
For example, to know the address where to find the fire data object (Figure
4.16) it is sufficient to add to the base address of the table in Figure 4.15, which
is
6BD1h
he
Fire type object identifier that is
0Ah
multiplied by
02h
(an address
requires two bytes), the result will be
6BE9h
. In the latter direction we find
defined data type object fire. Data defining objects
stored in the format shown in Figure 4.17.
Figure 4.17 - Data format of a sprite objects
The result will be one or more groups of 6 bytes as each group follows the format
Figure 4.17, this format defines:
41
Page 42
Byte ?
00
id contains the sprite.
? Bytes
01
,
02
Y
03
containing the X, Y and Z of the sprite.
Byte ?
04
It contains the flags to determine if the sprite requires rotation
horizontal or vertical, if an object movement and its dependence on
another object.

Byte ?
05
contains data for position corrections (see section
4.7).
There is a series of objects (moving on a regular basis, such as
guards) that require a position correction based on movement, this
It specified in byte
05
Figure 4.17. Contains if any of the coordinates
It requires correction.
Finally the example data, the fire (Figure 4.16) are:
0B5h
,
006h, 006h, 00Ch, 010H, 000h
,
000h
The end result in the working memory will be:
Figure 4.18 - Data from the fire type objects in working memory
We can see that there are more bytes in addition to the boxed in Figure 4.18,
corresponding to the four replicas of the fire object type. These data are used
play for other aspects of objects that do not interest us for us
display.
The function RetriveScreen places from memory address
5C88h
(See figure
4.20) all information in the room and, since then, is in that place where
They made all the changes that occur at the location while the player is
in her.
For each fund or object location (see Figure 4.20), we have a
at least two lines of groups of 8 bytes. A background or object can be represented
for over a sprite. Each represents a sprite with its position,
dimensions and information representation (see Figure 4.19).
42
Page 43
Figure 4.19 - Information of a sprite
Figure 4.20 - Total Dynamic Room
We can also see in Figure 4.20, finally what is stored in
Memory is the set of sprites that conform both the substance and the
43
Page 44
objects with information concerning the use of the technique filmation. How can we
see here, the information is not already compressed. In contrast, static memory
Fire type object is defined once and then enumerated the different
positions. Now, however, in the dynamic memory, the fire object has
four performances with their respective attributes specified separately.
4.6 SCREEN ELEMENTS
Once you placed all display elements in the working memory, and is
you can draw the screen. For example, the first sprite room
0Eh
is the
01h
(See figure 4.7), which corresponds to the sprite
02h
(Figure 4.21). This information has
It has been deposited in the working memory by consulting information
static. Program continuously checks the status of all objects
room and, if necessary, updates information regarding the position, state, etc.
But this update is performed only on the dynamic information stored
in the working memory. Therefore it is extremely important for this project
understand completely.
Figure 4.21 - First sprite room 00Eh
For information graphically represents the sprite, the program
see the table located from the position

7112h
containing pointers to
each sprite data (Figure 4-22):
Figure 4.22 - Table data pointer sprites
The identifier for each sprite (in Figure 4.21 is 02h) multiplied by
02h
(As a memory address requires two bytes for storage) serves
to add it to the start address of the table in Figure 4.22 and get a pointer
in the direction where textures are stored they represent a sprite,
7112h +
44
Page 45
04h = 7116h
. In this direction we will have the data to account
graphically a sprite.
The data that stores a sprite follow the format shown in Figure 4.23.
Figure 4.23 - Storage of graphics data of a sprite
Defined as:
Byte ?
00
indicates the width of the graph.
Byte ?
01
indicates the height of the chart.
? From byte
01
We have graphical information.
The product of the width and height values ??indicates the total number of bytes
will form the graphic. Thereafter, the program will simply taking the
successive bytes in pairs, correspond to the first texture image and
second texture representing the opaque mask. It reads up to obtain the
total number of bytes that make up the graph. It will be only draw this information
taking into account the size and position (Figure 4.19).
Figure 4.24 - Image and mask
Section 4.9 shows a complete example of the data of a sprite.
4. 5
Page 46
4.7 3D 3D WORLD HIGH AND LOW
One of the characteristics that should be taken into account was that, although the
coordinate of funds, and the main character objects were initially in
3D low-resolution (low) in the working memory are stored in 3D high
resolution (high). That was like to graph them by Filmation technique. The
solution was in the document data format Knight Lore Christopher Jon Wild.
Using the following system of equations, we can go from 3D to 3D High and Low
vice versa, which was what we wanted, worked as the 3D dynamic information
High resolution. The system is as follows:
(
) (
)
(
) (
)
(
) (
)
Screenz
Z
Z
Z
Y
Y
Y
X

X
X
L
H
L
H
L
H
+
·
+
·
=
+
·
+
·
=
+
·
+
·
=
4
12
72
8
16
72
8
16
1
1
1
L
X
,
L
Y
Y
L
Z
are the coordinates in
D
3
low resolution (read from the information
static game).
1
1
, Y
X
Y
1
Z
they are values ??that form part of the information
defines the sprites and for working with objects traveling alone (see
Figure 4.17). Depending on each object has a value or another. Finally Screenz is
a constant that always pays 128.
With this system we could move from
L
D
3
to

H
D
3
and vice versa easily.
A detail of the information which we did not have was the point of reference screen. It was solved in a very simple way, since it was only necessary to go changing the values ??of an initial position of an object in a screen and go visualizing the changes.
Taking as an example the room 255 (figure 4.25) definition was modified to change the position of an object
02Eh
=> (6,5,0) to
000h
=> (0,0,0):
0FFh, 00Bh, 006h
002h, 003h, 00Ch, 0FFh, 02Bh,
02Eh
, 035h, 037h, 03Eh
by
0FFh, 00Bh, 006h
002h, 003h, 00Ch, 0FFh, 02Bh,
000h
, 035h, 037h, 03Eh
The result can be seen in Figure 4.25.
46
Page 47
Figure 4.25 - Changing the X and Y axes
The following amendment illustrate us about the Z axis, modifying the same byte
02Eh
=> (6,5,0) to
0FFh
=> (7,7,3):
0FFh, 00Bh, 006h
002h, 003h, 00Ch, 0FFh, 02Bh,
02Eh
, 035h, 037h, 03Eh
by
0FFh, 00Bh, 006h
002h, 003h, 00Ch, 0FFh, 02Bh,
0FFh
, 035h, 037h, 03Eh
The result is shown in Figure 4.26.
Figure 4.26 - Modification Z axis
47
Page 48
Having done all this, and it was clear as the coordinate system was oriented, as shown in Figure 4.27.
Figure 4.27 - axis orientation coordinates
4.8 THE MAIN CHARACTER
The main character did not have information on how stored during the game, so we are in a location without objects and after stop time (canceling the call to the function that handles the course of the day and night, which also stores information in working memory), captured working memory completely. We move the character to recapture working memory and so on. After contrasting the various catches of the working memory are the coordinates of the character:
48
Page 49
Figure 4.28 - Coordinates Character
The coordinates
80 80 80,
located in memory locations
5C09h

,
5C0Ah
Y
5C0Bh
,
They represent the position of the body of the main character and
80 80 8C
, Located
in the memory locations
5C20h
,
5C2Ah
Y
5C2Bh
,
It represents the position of
head. The representation of the characters in the game consists of two blocks (Figure 4.29), in order to apply different animations in the same sprite separately.
Figure 4.29 - Sprites forming the main character
But with its coordinates it was not enough, as we needed some indicator to communicate the direction in which the character was oriented. It follows controlling the following bytes marked in Figure 4.30.
Figure 4.30 - character oriented data
The bytes located at the address
5C41h
Y
5C45h
(Figure 4.30) indicate the sprite
character (head and the body). This is not enough, because the sprite with
49
Page 50
orientation north of the main character is the same as the west facing but horizontally inverted, just as the sprite is facing south investment horizontally-oriented east sprite. Here come into play bytes of address
5C0Fh
Y
5C2Fh
(Figure 4.30). Observing these bytes discovered that serve to differentiate sprites having the same identifier but the graphics engine reversed, as they take a different value depending on whether the sprite appears inverted or not.
We can see this in more detail in section 5.6.
4.9 GRAPHICS AND OBJECTS OF FUNDS
As mentioned in paragraph 4.6, the graphs representing the sprites funds and objects, are located in memory and use a special format for storage (see Figure 4.23). Then we see how the "read and interprets "the visual appearance of an object more represented in the game, block, which we can see in Figure 4.30.
Figure 4.31 - A block
There are many replicas of this object during the game, including several objects block with subtle behavioral differences, which are represented with the same sprite. The object of figure 4.31 specifically, is represented by the sprite
07h
.
The information graphically defines the sprite block shown in Figure
4.32.
Figure 4.32 - Graphic information sprite block
And to understand the format shown in Figure 4.33.
50
Page 51
Figure 4.33 - format graphic information of sprites
The program, to acquire the graphic information of a sprite using the pattern

graphic information sprite Figure 4.33, applying the following steps:
• you obtain the first byte corresponds to the width of the sprite. Looking at the figure 4.32 we see that in the case of the block has a value of
04h
. This indicates
This sprite "measured" 32 pixels wide, it is because each byte
It represents 8 pixels: 4 x 8 = 32.
• The second byte corresponds to the height of the sprite. In the case of Figure 4.32 is
1Ch
28 decimal. This indicates that the sprite will block up to 28
pixels (here is not multiplied by 8 for a byte represents only a row, not a column).
• Proceeds from the two previous bytes we get the total size of the sprite,
4 bytes x 28 bytes = 112 bytes. This result indicates that the information
sprite graphics consist of 224 bytes: as we can see in Figure
4.33 information follow the pattern of image - mask, therefore the image
It will consist of 112 bytes and 112 bytes mask.
• Now the process is acquiring the bytes go on until
the total number (224). In our project we have discarded the mask as there
It served us all. The first byte of the image decompose
as shown in the following table:
BYTE IMAGE
BINARY
PIXELS
00h
00000000b
01h
00000001b
80h
10000000b
...
...
...
07h
00000111b
...
...
...
Once acquired all bytes stored in the correct structure, it can be
used to generate textures. All funds and graphics objects follow the
same process for purchase.
51
Page 52
52
Page 53
5. IMPLEMENTATION
2006 KNIGHT LORE
53
Page 54
5.1 POINT OF ENTRY
Once before the emulator source code, we set out to modify it
to allow the coexistence of the two windows (the emulator with the classic game and 3D environment).
To create the entry point of the new 3D environment had to encapsulate much
the main loop of the emulator in a specific function, so we could
call flow control and multiplexing and execution. We create a function that
He is performing an iteration of the emulator and then add a feature that was responsible
of refreshing the new 3D environment.
Figure 5.1 - Diagram of initial entry
It reached this point a problem arose when we pass control to the new environment
3D. By definition, OpenGL is a state machine, and once happened the

control iterates indefinitely until the program ends or is forced to
end up. This caused the emulator to stay "stuck" running the 3D environment.
The situation brought a change of approach to the modifications of the emulator.
Instead of running the function performed by an iteration of the emulator, and then run
which generates the 3D environment, we decided to grant full control to the 3D environment and
by own OpenGL functions that let you specify what to do in times
idle, giving iterations to go emulation function.
Figure 5.2 - Diagram of current entry
In Figure 5.3 we can see how the scheme was implemented Figure 5.2
to allow the simultaneity of the two environments. The main function of emulator is
which it is called main (or emuMain, depending on the compiler). We can see
I commented on how the emulator originally functioned code: preparing a
loop was performing successive operations, which are now encapsulated in the
emuMainLoop feature, allowing call when deemed necessary to run
a "cycle" of the emulator.
54
Page 55
The code continues with a series of functions GLUT (carrying the glut prefix) that
OpenGL charge of defining the environment, of which two are responsible flow execution:
glutDisplayFunc ()
Y
glutIdleFunc ()
.
The function
glutDisplayFunc ()
it addresses indicate the function performed drawn,
it will function
display ()
. Now whenever the user changes the window, or
generates an event the function is executed
display ()
that is in charge of call
function data acquisition and, having done that, translate them
graphically. The fact that the function
display ()
is associated with changes
may occur does not mean that it can also be explicitly called.
The function
glutIdleFunc ()
handles will specify that in periods of
Inactivity in this case function call
emuMainLoop ()
.
We see that these two provisions make it go drawing the new environment
when necessary, and also the code presents emulator that runs the
Figure 5.3. Originally, the function of Figure 5.3, consisted only of the loop, and within
it all statements which currently consists of the function
emuMainLoop ()
,
you are judgments are what make possible the operation of the emulator.
Figure 5.3 - Implementation of point of entry
55
Page 56
After implementing the pattern, they were seen on two environments at once,
as it is shown in Figure 5.4.
Figure 5.4 - The two environments running simultaneously
5.2 DATA ACQUISITION
First it was necessary to create a function to acquire all graphs
as may be required for the new recreation of the game. This function is called

cargar_datos_habitación ()
indicia and as its name, is in charge of acquiring
game information, specifically the sources that consultation are:
? Information defining initial static locations
? Working memory
? Table Data of funds
Table ? data objects
Before anything was added to the emulator code structure to store
acquired data, whose pseudocode is presented in Figure 5.5:
Figure 5.5 - Data structure of the room
56
Page 57
The pseudocode of the function
cargar_datos_habitación ()
It is found
in Figure 5.6:
Figure 5.6 - Pseudo function
cargar_datos_habitación ()
Room attributes are acquired from the memory locations that
they contain:
? The identifier of the room (with a value between
000h
Y
0FFh
) is
located in the direction
5C10h
.
? The color of the room is in the direction
5BADh.
? The size (x, y, z) can be found in the directions
5BABh
,
5BACh
Y
5BAEh
respectively.
Room funds acquired static information. The reason
it takes only this place is because their data never vary. The
walls or doors never move or disappear. The portion shown
code that performs this in Figure 5.7, which reads as often as funds are available,
the
IDs of funds. To fully understand the code in Figure 5.7
please consult section 4.5.
Figure 5.7 - Data collection of funds from the static information
The data obtained from the objects must be purchased not only information
static, but also of working memory and the data table objects.
Static information, we get what objects have the room and the number
their representations (see Figure 5.8).
Working memory, we acquired coordinates (objects can be moved or
disappear) and other data (see figure 5.9). Finally table object data
to obtain the values, if needed, positioning corrections
(See section 4.7). To fully understand the code in Figure 5.7 and 5.8
can consult section 4.5.2.
57
Page 58
Figure 5.8 - Obtaining data objects static information
Figure 5.9 - Preparation of the data objects in the working memory
OpenGL 5.3 INITIALIZING
OpenGL is a library that works as a state machine. When the
use, first thing to do is enable and disable options, make
certain actions that will result a screen representation of a series of
data depending on the state in which we find ourselves.

This first project is called an initialization function that
responsible for defining color attributes, projection and Z-Buffer (array that stores
depth information that allows overlapping objects in the correct order).
58
Page 59
Figure 5.10 - OpenGL initialization function
Subsequently, whenever you enter the function that plots the new screen,
just after loading the data of a room, the following functions are executed
OpenGL.
Figure 5.11 - functions that define, position and orient the camera
Its basic utility is tell OpenGL where "looks" at the camera. It is included in the
drawing function screen 2006 Knight Lore, because anytime
the user can change the perspective of the game.
5.4 DRAWING FUNDS
The walls of each room were drawn using textures from the original game.
The process involved the capture of each sprite (see Figure 5.12), which formed each
wall for storage in an image file. In section 4.9,
It explains in detail how the game's graphics are stored.
Figure 5.12 - Different sprites representing walls
59
Page 60
The funds drawn function examines the dynamic data of the room, to
get the identifiers of the sprites that will be part of the wall. Then loads
files corresponding to the sprites wall and positioned according to their
position in the new environment.
Keep in mind that the captured textures from the original game feature
It spells perspective distortion, but that was solved thanks to which, by
OpenGL, when you apply a texture to an object can correct their
Dewarping by positioning technique.
Figure 5.13 - Fixed texture perspective through technical Dewarping
The technique Dewarping choose a number of points, which are passed as coordinates
OpenGL texture. Then the latter, makes the necessary corrections to apply
texture for proper display, conceptually shown in Figure 5.13.
The remaining funds (mainly doors) will be drawn like any other
object primitives using OpenGL.
5.5 DRAWING OBJECTS
All objects are drawn using OpenGL primitive, an example we can
observe in Figure 5.14. The function that takes care of drawing is called by the
main function objects as many times as the location has. Each call
It includes the arguments necessary to describe the object. Depending ID
receiving object, applies a primitive other. If an object is not defined
then the default object, which in our case is a small sphere is drawn.
60
Page 61
Figure 5.14 - Room with objects
Note that although there are several objects that, although they share a look
visual, facing the interaction in the game behave differently and
even they do not have the same identifier sprite. It is the case of an object very
common block (see Figure 5.15), which have different entities (fixed block,
block disappears when stepped on, block movement), although internally
the game treats differently, their graphing requires no such differentiation and
He is considered and drawn as if they were a single object.
Figure 5.15 - Different sprites with the same image
The objects that have animation (ball, fire, soldier, etc.) require
different sprites to represent it. Also in this case it is somewhat transparent to the
new game environment and it is the original game that is changing memory
depending on the work required for each animated sprite depending on their status
object.
Figure 5.16 - Sprites representing various animations
61
Page 62
5.6 DRAWING THE MAIN CHARACTER
The function draws the main character, first you get the coordinates

(X, y, z) located in memory locations
5C09h
,
5C0Ah
Y
5C0Bh
(See section 4.8).
Then converts these coordinate values ??to OpenGL 3D High setting (see
4.7). Once we know the position the next step is to deduce the state of
character (male or wolf) and orientation. Querying the memory
5C41h
obtain the ID of the sprite character's body, it lets us know
their status and part of their orientation. To finish knowing it, we consult
value of the memory location
5C0Fh
Depending on the result we can deduce
definitely guidance.
Figure 5.17 - Code to get the position and orientation of the main character
In Figure 5.17 we can see how the function gets the first
coordinates of the working memory and I rolled converts OpenGL. Next
You obtain the byte located in the memory address
5C41h
to discern if the character
it is man or wolf (see Figure 5.18). Finally according to the value of the position
memory
5C0Fh
and sprite determine the orientation (see section 4.8).
62
Page 63
Figure 5.18 - Sprites main character
Once acquired data we require the character to draw, apply
first rotation, then a translation and then used several
OpenGL primitives to finally draw the character. In these operations was
taking into account the guidance (for that rotation is applied) and the coordinates of
its
position, for proper positioning.
Figure 5.19 - Main character in the old and new context
63
Page 64
64
Page 65
6. RESULTS, CONCLUSIONS
AND FUTURE WORK
65
Page 66
6.1 RESULTS
The end result consists of a new environment, as seen in Figure
6.1. Figure 6.2 shows a sequence of motion of a bouncing ball. Also
you have the option to view the game in 3D from different perspectives and
improved visual appearance, as we saw in Figures 6.3 and 6.4.
Figure 6.1 - Final appearance of the hybrid application
Figure 6.2 - Sequence of a ball moving in a room
66
Page 67
Figure 6.3 - Different views of a room of Knight Lore 2006 and the original
Figure 6.4 - Different views of a room of Knight Lore 2006 and the original
67
Page 68
6.2 CONCLUSIONS
We have succeeded in developing a remake of a classic ZX Spectrum game, but
without rebuilding the game completely but add functionality to generate a
new graphics context, being diverted flow chart and implementing it again
3D in another context. So we've studied the architecture of the ZX Spectrum, made

use of reverse engineering to figure out how the game works and graphic level created a new rendering engine based on geometric elements and textures.
On a personal level, we conclude the work satisfied by the results, which to the objectives specified at the beginning, allowing gird have a functional platform to add new features and improvements.

6.3 FUTURE WORK

Regarding future work to develop could be:
? Draw objects using different techniques appropriate to represent and get a better finish thereof.
? Apply light sources to enhance the look consistent to the theme of game.
? Improve the performance capture sprites to be obtained directly from the code game.
? Generalization of code to other games that work suits by the same techniques, although various games removed is reduced by the same company and related industries.
68
Page 69
69
Page 70

7 REFERENCES

70
Page 71
The Z-80 microprocessor
Primitive Francisco
http://www.microhobby.org/varios/MICROHOBBY-ElmicroprocesadorZ80.pdf
Course Code Machine ZX-Spectrum
Jesus Alonso Rodriguez
http://www.microhobby.org/varios/cursocodigomaquina.zip
Filmation initiation system
Magazine MicroHobby 96, 97, 98 and 99
http://www.microhoby.org
On filmation
Neil Walker
http://retrospec.sgn.net/users/nwalker/filmation/
Knight Lore data format
Chris Wild
http://www.icemark.com/dataformats/knightlore/index.html
Spectrum emulators
Santiago Romero
http://www.speccy.org/sromero/spectrum/emuls/
The OpenGL Programming Guide
Dave Shreiner, Mason Woo, Jackie Neider and Tom Davis
http://opengl.org/documentation/red_book/
The OpenGL Utility Toolkit (GLUT) Programming Interface
Mark J. Kilgard
http://www.opengl.org/documentation/specs/glut/
Introductory course OpenGL
Jorge Garcia Ochoa Aspuru
http://www.bardok.net/content.php?lang=1&article=2
Notes of OpenGL and GLUT
Cristina Morales Cañero
http://www.cvc.uab.es/shared/teach/a21306/doc/Apuntes%20de%20OpenGL.pdf
71
Page 72
Macedonia Classroom - Course OpenGL graphics programming
Oscar Garcia
http://usuarios.lycos.es/macedoniamagazine/opengl.htm
Computer Graphics practices
Arno Luzon Formilla and Maria Victoria Garcia
http://trevinca.ei.uvigo.es/formella/doc/ig02/
Computer Graphics
Inma Remolar Oscar Belmonte Ramos and Francisco J.

http://graficos.uji.es/grafica/
Infographics Practice I
Alejandro Ramirez Montero
http://www.ucbcba.edu.bo/maestrias/MATERIAS/grafismo/infografia/index.html
NeHe Productions - OpenGL Lessons
http://nehe.gamedev.net/
72
Page 73
APPENDIX
73
Page 74
A. FUNDS
ID
FIRST NAME
GRAPHIC
00h
Northern Arc
01h
Arco this
02h
Arco south
03h
West Arco
04h
Arc tree north
05h
Arco tree this
06h
South tree Arc
07h
Arco tree west
08h
North Reja
09h
Reja this
0Ah
South gate
0Bh
West gate
0ch
Room walls Size 1
74
Page 75
0Dh
Room walls Size 2
0Eh
Size 3 room walls
0Fh
Tree Room 1
10h
Fill grove west
11h
Fill Grove North
12h
Sorcerer
13h
Cauldron
14h
This high arc
15h
High arch south
16h
Arch this high basis

17h
High arch south basis
75
Page 76
B. OBJECTS
ID
FIRST NAME
GRAPHIC
00h
Block
01h
Fire
02h
Ball with vertical movement
03h
Rock
04h
Gargoyle
05h
Block skewers
06h
Pushable trunk
07h
Pushable table
08h
Guard moving east - west
09h
Ghost
0Ah
Fire movement north - south
0Bh
High block
0ch
Ball with vertical movement
0Dh
Guard patrol perimeter
0Eh
Movable block east - west
0Fh
Movable block north - south
10h
Movable block
11h
Block high skewers
12h
Falling spiked ball
13h
Spiked ball falling from the ceiling
14h
Fire moving east - west
15h
Draggable block
16h
Immovable block
17h
Random ball movement
18h
Ball falling from above
19h
Spell murderer
1ah
Grille with vertical movement
1bh

Grille with vertical movement
1Ch
Ball with vertical movement
76