OPERATION ALEXANDRA

A 64Kb video game for Amstrad CPC developed by

Code & Engine	JAVIER GARCÍA NAVARRO
Gfx & Game Design	RAFA CASTILLO
Music & Fx	JOHN MCKLAIN
Testing	Blackmores
	Metr81
	JGonza
	Pomez666

- MAKING OF -

v1.0 (24/10/2018)

1. THE ORIGIN

It all started shortly after the end of CPCRetroDev in 2017...

JGNavarro was "*on fire*" with GameMaker after finishing the conversion of Profanation 2 from Amstrad CPC to Windows and Android, and thought it would be a good idea to develop a new conversion of a videogame where all the graphic and sound material was already finished. Through McKlain, launched to Azicuetano the irresistible proposal to make a similar conversion with Baba's Palace, and in less than 2 weeks already had ready and published the conversion for Windows and Android. An intense work but very gratifying for all.

From this collaboration, the contacts between the all were more and more frequent, culminating with the incorporation of Azicuetano to the 4MHz team, the group where JGNavarro and McKlain together with Sad and LordFred develop homebrew projects for Amstrad CPC.

2. THE IDEA

Already by November 2017, Azicuetano was several weeks behind a good storyline for a video game. The basic idea was the only thing that was clear enough: a shooting video game. I had also developed some concepts of characters doing some actions, like walking and shooting... but I was aware that without a good story you could not start a development in conditions.

It is vital to have a good story as a starting point, because it is precisely the story that allows us to evaluate and contrast the coherence of all the ideas and creativities that come later.

During the search for that story as it came to a curious news of 2016. A shocking news that gave rise to what would be the story on which the video game Operation Alexandra would be developed:

Secret German World War II Base Rediscovered Near North Pole

This news was the perfect starting point to develop a great fictional story on which our video game would be based:

> *During World War II, the Nazis built this base on the island Alexandra Land, within the Arctic Circle, with the aim of investigating a rugged UFO and get a technology that would give them a great advantage in the war. They built the base on it and organized a complex excavation to gain access to the interior of the mysterious artifact. But the way they opened to enter the base allowed the liberation of some fearsome beings that in a short time killed all the inhabitants of the base.*
>
> *The base remained forgotten for 30 years until in the mid-1970s, the protagonist of the video game - a Soviet army officer - discovered it by chance and decided to put an end to all the horrors that raged through the old base before they ever reached the outside.*

Azicuetano initiated the design document of the videogame, and there he detailed the multiple references to the news in different media, together with a brief fiction story, to present the project to the rest of his colleagues in 4MHz. Finally, the idea managed to excite the rest of the group, so that a work team composed of JGNavarro, Azicuetano and McKlain was organized to tackle the project.

3. THE PROTOTYPE

The technology base of this development would be La Hormigonera, that is, the engine in ASM that JGNavarro has in continuous evolution to develop 4MHz video games.

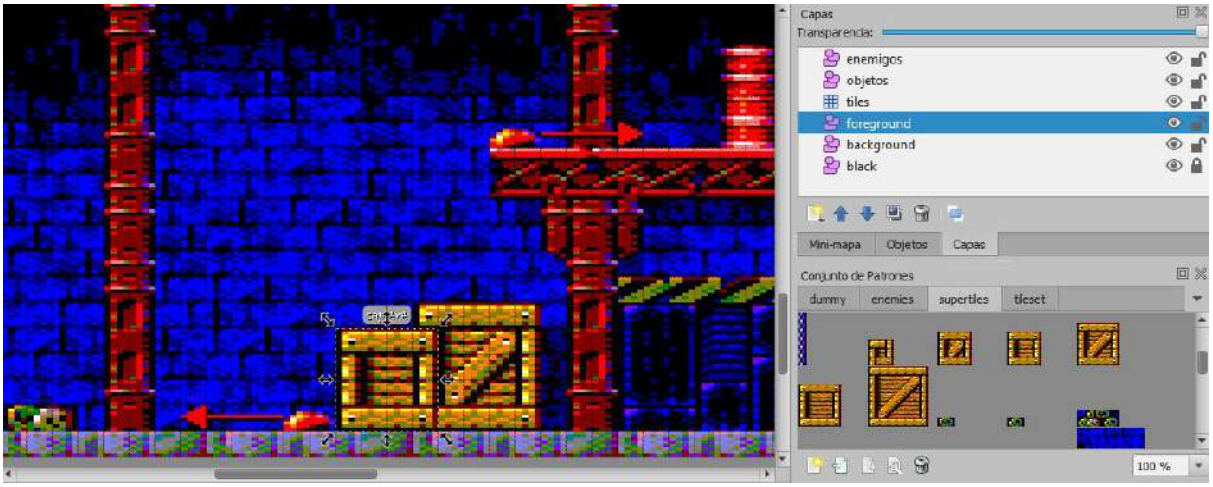
In Profanation 2, the previous 4MHz project, JGNavarro and LordFred worked with a system based on tiles and supertiles (a group of tiles) to create the definition of the scenarios. It is a complex system, but it has 2 fundamental advantages:

1. Save bytes in the definition of each screen.
2. A great graphic richness, since it allows to work with 4x8 tiles.

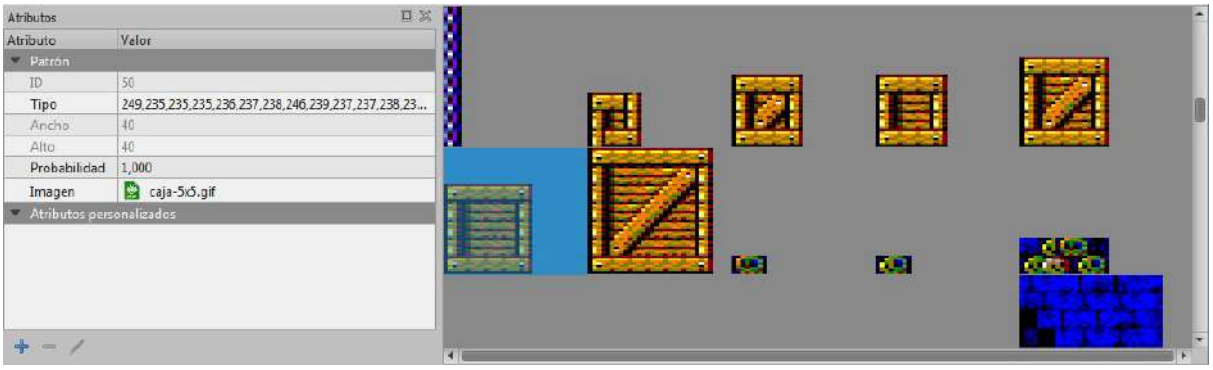
In Profanation 2 this system worked in a masterly way, and partly, thanks to it, they managed to adapt to 64kb. the initial development destined to 128kb. In that moment, the work against the clock prevented to organize a good system of work, and all the task of coding screens ended up being a really hard manual work.

Now we had the opportunity to automate this task in order to reduce errors, save time and be able to dedicate effort to other important parts of the development. We also took advantage of the moment to introduce improvements that would save more memory in the screen definitions. Thus, the beginning of the project consisted of developing a prototype with an automated system for the creation of screens based on tiles and supertiles.

As a basic tool for the creation of screens, Tiled was used, but in an unconventional way. Instead of creating screens based on tiles, they were created based on supertiles. These supertiles consist of tiled "patterns". To create these patterns dummy graphics have been used that recreate all the supertiles we use in the game.



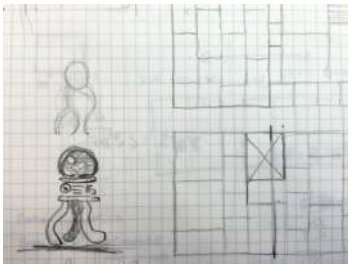
Within each tiled pattern we have included the definition of each supertile. Thanks to that, and to the meta-information that Tiled generates, we have in each .TMX file all the information we need to compose our screens.

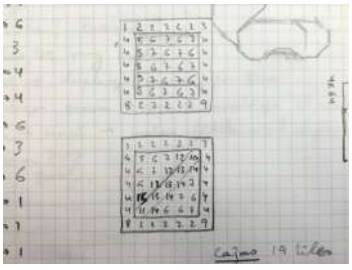


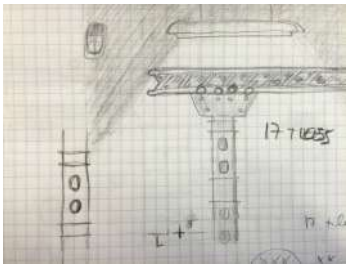
JGNavarro developed a tool in Python to parse the .TMX files and perform an automatic translation of each screen to assembly code for Amstrad CPC.

Pro-Motion(Cosmigo) was used to design the tiles and supertiles. This tool has great advantages, such as the possibility of working with anamorphic pixel 2:1 (the pixel width of Amstrad mode 0), specific management of tiles and the possibility of exporting tiles structures to easily generate the definitions of supertiles.

After several weeks of work and tests we were able to establish and well adjusted the working method, and we developed our first prototype: 3 screens linked together, with the protagonist and a couple of enemies:







4. THE DEVELOPMENT

With the prototype working we were able to make the first estimates. A spreadsheet was created to make all the memory estimates, and there was established how much memory would be assigned to tiles, to the definition of supertiles, to the definition of screens, sprite protagonist, enemies, character set, artificial intelligence, logic, etc ....

Once the memory forecast was established, we began to develop the bestiary: the enemies and their behavior. This point was fundamental, because thanks to having well defined the set of enemies, with their characteristics, behavior, AI, etc.. an intelligent level design can be approached.

In several cases, prototypes were developed as animations, in order to study whether the behaviour we had devised fit in with what we were looking for and, by the way, to serve as a reference for programming.

Once the bestiary was closed, the design of levels began. At this time, the creation of the screens consisted of a perfectly established procedure, so the difficulty lay primarily in the creativity and technique used to achieve savings in bytes and speed in painting.

The design of levels required a considerable amount of time, so that this activity was carried out in parallel to the implementation of the functionalities required by the game: special effects, triggers and Interactions with objects, dialogue system, door opening, power-ups, main menu, contextual menu during the game, etc.

By the time half of the screens that make up the game were designed, almost all of the game's extra functionalities were already implemented, so we considered it was the ideal moment to start the beta-testing phase.

In the selection of beta-testers we looked for different profiles with complementary skills. In this case we opted for a very technical profile, able not only to find bugs, but also to test and test until we found out the modus operandi to reproduce it. Another advanced player profile that could guarantee that the game can be completed from start to finish. And finally, a "regular" player profile, which could give us feedback on how difficult most players will be able to experience.

A lot of adjustments were made after the evaluation of the beta-testers team's reports, categorized into difficulty adjustments and bug fixes. Meanwhile, the task of level design came to an end.

The fight against the final enemy and ending was implemented, and a new beta was supplied to the testing team. The new reports provided, after testing with this second beta, indicated that the improvements in difficulty implemented had worked well, and that the

overall rating on the game was very good. So we considered everything related to the development of the game closed.

At this point, we only had about 150 bytes, needed for the stack, so there was no more memory to implement anything else. We were very pleased to have managed to meet the initial forecast, including everything we had planned: screens, enemies, melodies, etc..

...although we missed a little intro. And here it was during an improvised "brainstorming" when we had a spectacular idea to include the intro: to compact ALL the game and get enough space for an intro that would be seen when loading the game, and that after its viewing would be over-written when decompacting the game over all the available memory.

After several tests JGNavarro worked the miracle: 17kb to our disposition to create an intro that allowed to put the player in situation. A real pass!

So we got to work with the intro, but without much time ahead, since these tasks were not contemplated.

After incorporating the intro, we went through a process of creating Release Candidate and testing, which came to an end with Release Candidate 5. We already had the game ready to deliver!

5. CREATIVITY, DESIGN, GRAPHICS AND MUSIC.

Operation Alexandra is inspired by a multitude of works. From films like "The Thing from Another World" (1951, Dir. Christian Nyby and Howard Hawks) to video games like The Sacred Armour of Antiriad (1986, Palace Software), and melodies like the famous choirs of the Russian army.

5.1 Graphics

Creating screens from supertiles involves meticulous design work, but the quality of the result, compared to the small amount of information needed for each screen, makes it well worth it.

Making bad decisions when it comes to composing a screen can make it double or triple the size. You have to be very conscientious in the creation of supertiles, since you need a varied repertoire to be able to have flexibility when designing a screen.

That flexibility is fundamental, because the most important thing about design is not aesthetics, but functionality. Let it be fun, come on! You can't see the level design ballasted because you don't have the "piece" you need...

The developed system of supertiles allows overlapping, which is sensational to create very attractive screens, since it is not necessary that the different structures created with supertiles fit perfectly with each other, achieving depth and breaking the sensation of repetition.

For the graphic development there has been a main tileset composed of 256 4x8 tiles, and a secondary tileset composed of 192 4x8 tiles. The main tileset is used to create all the game scenarios, and the secondary tileset for the rest of the graphics needed, such as the font, the HUD, main menu, objects, the final enemy,...

There have been developed a total of 230 stiles of very diverse sizes, and whose definition occupies about 4kb.

Aseprite has been the tool used for the development of all the animations: protagonist, enemies and explosions...

5.2 Bestiary

The development of enemies has been one of the most fun tasks. Devising their behaviour, attack, way of behaving, etc. would later condition the design of screens.

Broadly speaking, the most relevant details in the development of enemies have been:

1. The ability to shoot.
2. The enemies that shoot must be predictable.
3. When an enemy is vulnerable or not.

The shooting of enemies is directed towards the exact position of the player. Undoubtedly, a trait that denotes intelligence.

We have opted for a system in which enemies do not shoot fortuitously, so that enemies who shoot present 2 behaviors that allow predicting their shooting:

1. Characteristic gesture, which allows the player to infer that the enemy is about to shoot.
2. Shooting with a certain cadence, which allows the player to predict when he will make his next shot.

On the other hand, certain enemies can change their behavior on receiving an impact, so that when you shoot an enemy, when you mess with him, they react and can make variations in their displacement that can catch the player off guard.

There are immortal enemies, others that close and become invulnerable, others capable of jumping between 3 and 11 times their height, another can shoot 2 bullets with parabolic shot, others to die leave on screen part of their mechanism of displacement and the player can interact with him and climb on top, others are spinning to announce his imminent shot ...

5.3 Gameplay and level design

The level design has been a great challenge for many reasons. And is that along with the control of the protagonist and the behavior of enemies is one of the biggest responsible for the gameplay of the video game.

With Operation Alexandra, we weren't looking for a shooting game where the protagonist advances by destroying everything that is found, but rather a game where the player had to put into practice a certain strategy when tackling each screen. You have to find a way to shoot enemies, but also run away at the right time. There are areas from where you can approach the attack on enemies with better guarantees, and that you know how to find them. At certain times, the player can choose between different paths and not all are equally suitable depending on the point of development of the game.

The game consists of a series of visual effects to convey well to the player that is happening at all times. For example, the protagonist flinches, gets white ink and gets a few moments of immunity when he receives damage. When enemies receive damage, they are also dyed white and a specific fx sounds. This detail is important because not every time an enemy is hit by a bullet does he suffer damage.
Of course, the jumps are an important element, and for this it was essential that the control of the character was completely accurate, as it has been achieved.

Generally, at the point of access to a screen the player is usually protected, or protected for a reasonable time. In this way, the player always has time to think and plot his strategy.

This videogame has a small adventure component. In order to move forward you have to perform certain actions such as picking up and using objects in certain places, or removing certain obstacles. To help in these actions the player has the protagonist's inner monologue, which offers clues on the actions to be performed at certain points in the game.

The complete mapping is arranged in such a way that, from any screen, the player can go to any other screen (once the player has released the accesses, of course). That is to say, there are no points from which it is not possible to go back. In addition, sometimes there are several paths for game after game, the player will learn which is the best course depending on the time, thus optimizing their time and reducing the risk of being injured.

A lot of work has been done to adjust the difficulty. At this point, the work of the testers is fundamental, as the developers easily lose perspective due to the large number of tests they have to perform. The goal from the first moment is to get a game where the player is able to improve game after game, and can get to the end with the appropriate dedication.

All the details related to the creativity and conceptualization of Operation Alexandra have been collected in a design document. This design document has been the most important reference to define what the game itself is. In this document has been detailed absolutely everything related to the videogame: history, texts that compose the intro, values of graphic palettes, definition and behavior of enemies, dialogues, special effects, final fight, ending. Here it is reflected from where to find a certain sprite, up to the succession of offsets in Y to compose the jump of the protagonist. Everything is reflected, absolutely everything. Not in vain, the design document of Operation Alexandra is composed of 66 pages.

5.4 Visual effects

During the game, players will be able to experience certain visual effects that are not very common in developments for CPC:

5.4.1 Twilight effect

At the beginning of the game, the base is with emergency power and the generator is running out of fuel. The player will experience certain flickers in the light, the result of this lack of energy. If he descends to the basement, these screens will be in semi-darkness.

This effect has been achieved by changing the inks just after painting the HUD. A version of the palette has been developed by darkening all colours to give this feeling of gloom.

5.4.2 Snow effect

There are screens at the top of the base where you can see the outside, and where you can see how on the outside there is a great snow storm.

Some tiles have been generated with very particular colours in order to give the sensation of blizzard by means of successive palette changes. In addition, the colour changes take place in such a way that a parallel sensation is transmitted when working with flakes that move at three different speeds.

As this is an effect achieved by changing inks, this animation can cover a large part of the screen without affecting performance.

5.4.3 Beam effect

On the most advanced screens, in the UFO area, the player is in contact with alien technology. There are some rays capable of damaging the protagonist if he comes into contact with them. These rays emit a glow in the form of a vibration.

This glow is achieved by sacrificing an ink from the palette. This ink in the even frames is orange and in the odd ones yellow, obtaining this way the effect of vibration in the ray.

5.4.1 Flash effect

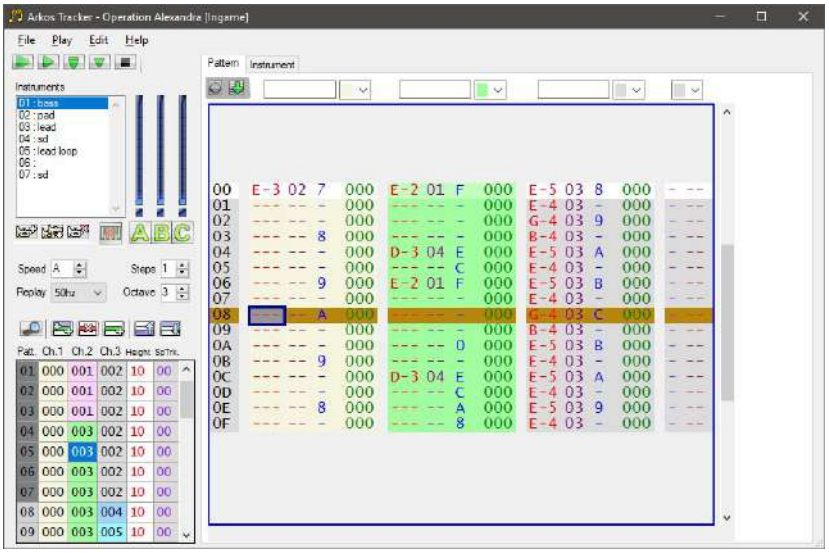
During the final fight, when the boss is damaged, a glow floods the screen.

Just like the penumbra effect, this flash effect consists of a change to a version of the clearer palette for 2 frames.

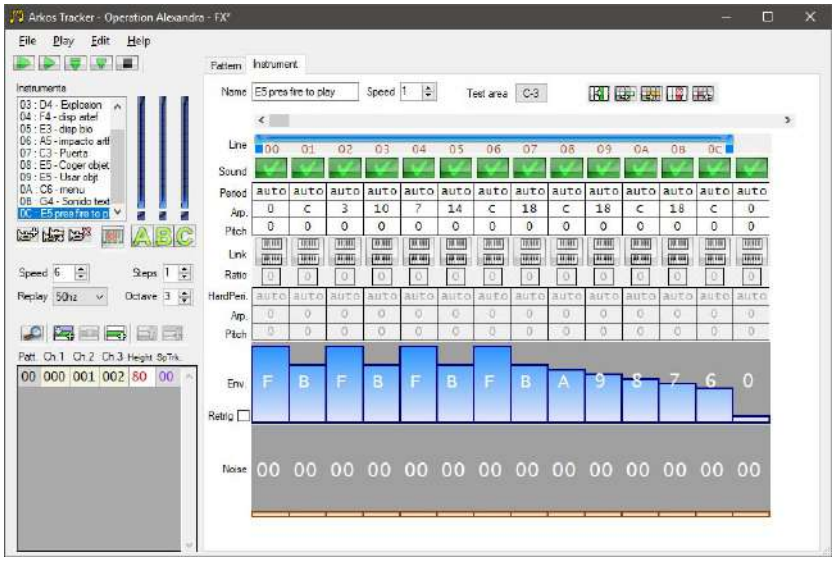
5.5 Music and sound effects

The inspiration for ingame music comes from the soundtracks made with synthesizer in the eighties in horror and mystery movies, mainly thinking of the music of John Carpenter. The idea was to accompany the player during the adventure and immerse him in the atmosphere of mystery and danger that surrounds the game, but without distracting him from the main action. This melody lasts 1m 55s and occupies 699 bytes. The music on the menu follows a different path and is inspired by the classic melodies played by the choirs of the Russian army, in an attempt to evoke the "glorious days" of the USSR during which the game takes place. This music lasts 23s and occupies 717 bytes.

The melody of the intro is in line with ingame music, a synthetic theme to set the dialogue between the protagonist and his classmates at the base. This melody only uses two channels to leave a free channel for the dialogues FX, which plays almost continuously. The idea of the FX is to simulate a conversation between the characters by randomly varying the tone of the FX by printing each letter of the dialogue and leaving a pause between words. The melody lasts 38s and occupies 425 bytes and the dialog FX occupies 62 bytes.



Entering fully into the section of sound effects, starting from a set of 12 unique sound effects we get to have 19 different effects reusing some base sound effects that are released with different parameters. For example, the effects of jumping, landing and impact of an enemy with our protagonist share the same base effect, but are launched with different note values, with the consequent saving of memory.



In the list of sound effects we have jump and fall effects, shots, impacts, explosions, short melodies to indicate the collection and use of objects and a fx for the dialogues of the hero.

Order	FX Nº	Name	Value	Chanel	Note
00	**1**	Shot hero	**36**	1	C3
01	**2**	Jump	**35**	0	B2
02	**2**	Landing	**26**	0	D2
03	**3**	Small explosion	**33**	1	A2
04	**3**	Big explosion	**50**	1	D4
05	**4**	Shot Artefact	**53**	1	F4
06	**5**	Shot Biologic	**41**	1	F3
07	**6**	Artefact impact (closed)	**69**	1	A5
08	**6**	Artefact impact (open)	**64**	1	E5
09	**5**	imp. shot with bio	**29**	1	F2
10	**4**	imp. shot with hero	**33**	1	A2
11	**2**	imp. enemy with hero	**45**	0	A3
12	**7**	Open door	**36**	1	C3
13	**8**	Pick up object	**64**	2	E5
14	**9**	Use object	**64**	2	E5
15	**10**	menu - esc	**72**	0	C6
16	**10**	menu - option	**65**	0	F5
17	**11**	Text sound	**55 (55-58)**	0	G4
18	**12**	Press fire to play	**64**	0	E5
19	**1**	Shot hero advanced	**41**	1	F3

The effects that use the noise channel are sent to the central channel to avoid conflicts in the use of the noise generator (the ingame music uses noise for the percussive sounds of the central channel), the fx related to objects are sent to the right channel to cut the main melody to sonar and the rest of fx are sent to the left channel. In total this FX game occupies 329 bytes in memory.

The music and effects have been made with Arkos Tracker 1.0, already a standard in the new productions for Amstrad CPC.

6. TECHNOLOGY

The programming has been completely developed in assembly language. Although at the beginning of the project many routines start from the base of La Hormigonera, during the development many of those routines are improved and optimized to solve in a more efficient way the particular cases that occur in this videogame.

In particular, there are several aspects that deserve a more detailed explanation:

6.1 Artificial Intelligence

During the design phase very particular and specific behaviours were established for each enemy.

6.1.1 Shooting to a specific coordinate

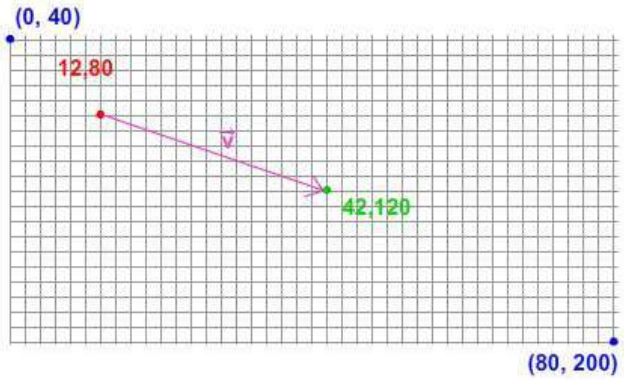
Of all the behaviors of enemies, the most relevant is the ability to fire a projectile at a specific point on the screen: where the protagonist is located. A trivial task at present, but certainly complex in 8bits.

To implement this action we developed a version of the Bresenham algorithm, and it really worked, but we were not happy with the result. The reason for this was that this algorithm is designed to paint lines between two coordinates so it forced to paint the bullet always in contiguous pixels. This caused the projectile to move 4 times faster on the X-axis than on the Y-axis. It wasn't bad, but we wanted to do better.

So we opted for a variation of one of the traditional algorithms to move objects between two coordinates. We improved the base algorithm called "Evaristo" to correct the anamorphic pixel of the amstrad 2:1, and the fact that it is naturally painted in exact bytes (2 in 2 pixels horizontally).

That is, we had to make a compensation of 4:1 in horizontal:

SQUARE PIXEL 1:1



$\vec{v}=(dX, dY)$
dX = x2-x1 = 42-12 = 30;
dY = y2-y1 = 120-80 = 40;

$\vec{v}=(30, 40)$

$|\vec{v}| = \sqrt{30^2 + 40^2} = 50$

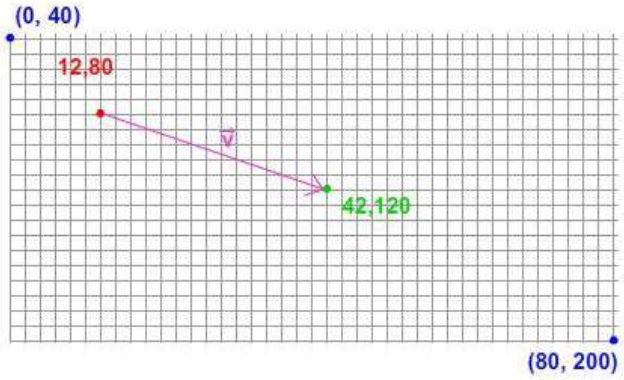
$\vec{u} = \frac{1}{50} \cdot (30, 40)$ --> $\vec{u} = (\frac{30}{50}, \frac{40}{50})$

$\vec{u} = (0.6, 0.8)$

IncX = velocidad * 0.6
IncY = velocidad * 0.8

IncX = 0.6
IncY = 0.8

WITH ANAMORPHIC COMPENSATION 4:1



$\vec{v}=(dX, dY)$
dX = x2-x1 = 42-12 = 30 * 4 = 120;
dY = y2-y1 = 120-80 = 40;

$\vec{v}=(120, 40)$

$|\vec{v}| = \sqrt{120^2 + 40^2} = 126.49$

$\vec{u} = \frac{1}{126.49} \cdot (120, 40)$ --> $\vec{u} = (\frac{120}{126.49}, \frac{40}{126.49})$

$\vec{u} = (0.9487, 0.3162)$

IncX = velocidadX * 0.9487
IncY = velocidadY * 0.3162

IncX = 0.9487
IncY = 1.2648

COMPENSACIÓN ANAMÓRFICA
velocidadX = 1
velocidadY = 4

The implementation of this algorithm is also not trivial as it involves square roots and floating point operations. But after a day of trial and error we managed to solve this problem. It was very gratifying to be able to see on screen the projectiles moving at a constant speed regardless of their direction!

6.1.2 Reactions to the attack.

A kind of reaction of the enemies to the attack has been implemented.

In some cases, not always, an enemy reacts to a bullet impact by changing its trajectory of displacement. This does not always happen, in fact it happens few times, but with that we achieve our goal, which is that you can not rely on the trajectory described by an enemy, as a hit could vary and catch the player off guard.

6.1.3 Choosing the moment of the shot

There is one enemy that fires and another that jumps with a fixed cadence and that, as stated above, is so that their attack can be predictable. But there are other enemies who make their own decision about when to fire, such as the Clam, the Turret and the Plant.

These enemies warn that they are going to fire with a characteristic gesture, but the choice of when to fire is made by the enemy, and they are not subject to a preset cadence.

6.1.4 Direction Changes

We have an enemy that we have called Rana, which has prefixed routes. This enemy can decide when to initiate a change of direction depending on the actions of the player and a certain component of chance. The most common way to avoid this enemy is to jump it, so a change of direction at this time can ruin the player's intentions.

6.2 Supertiles

As we have already mentioned, the screens are generated from supertiles with the main objective of saving on screen definitions and achieving graphic richness.

Several optimizations have been developed that allow greater compression in particular cases thanks to the repetition of supertiles in horizontal, vertical and screen mode.
This system makes the definition of each screen variable. It depends on the tiles and supertiles used, which layer they are in, how many times they are repeated on that screen, how to define raster type repetitions ... To take advantage of the optimizations is important to have a good eye on the level design.

The average screen size in Operation Alexandra is 126 bytes. It is important to point out that a screen is made up of 40 tiles wide by 20 high, so that a screen made up of 800 independent tiles can be coded in 126 bytes.

The overlapping of tiles/supertiles is possible but dangerous, as it increases the screen generation time. You could define a screen with only 32 bytes, using 20 supertiles of 8x10 tiles, which would involve drawing 1600 tiles to compose it. The definition of the screen would be small, only 32 bytes, but the painting time would be unbearable.

A specific tool was developed in Python to translate the screens created from Tiled to ASM code, with the consequent saving in time, reduction of errors and flexibility. Thanks to this tool, incorporating a screen modification into the game consisted of a task of seconds.

6.3 Management of sprites by interruptions and dynamic buffers.

The greatest complexity of a game developed in 64Kb. is the lack of space and the great sacrifice of memory that supposes to use double buffer to make the sprites paintings without uncomfortable flickers. For this has been used to do everything in its time, so that everything is controlled. But how to know which moment is the current one if the CRTC of Amstrad does not indicate where the beam of electrons is in every moment. So easy, executing the processes in the interruptions that, previously, have been defined to start in the same moment always: when the electron beam is in the upper left part of the screen (only moment in which CRTC informs).

In this way the first interruption takes place at the top right of the screen, and the following ones every 10Hz approximately. This means that you have to take advantage to paint on screen at times when you know that the electron beam is not "on top".

In addition, since it was unnecessary to move the objects to 50fps if not that with 25fps the ideal speed was obtained, it was decided to paint every two screen refreshments sprites (the same as saying 25fps).

For this reason the following actions have been defined for each interruption:

Interrupt / Screen area	Step 0	Step 1
0 Upper screen area	**SPRITE HERO** save background screen	**ENEMY 1** Print Screen Background Move sprite Print Sprite on Screen
1 HUD	**SPRITE HERO** Print Sprite on Screen	**SHOTS** Captura Fondo Pantalla Imprime Sprite Pantalla
2 Upper screen game	Pallete Efects Keyboard reading Arkos Player	
3 Middle screen game	**ENEMY 5** Print Screen Background Move sprite Print Sprite on Screen	**ENEMY 3** Print Screen Background Move sprite Print Sprite on Screen
4 Lower screen game	**ENEMY 2** Print Screen Background Move sprite Print Sprite on Screen	**ENEMY 4** Print Screen Background Move sprite Print Sprite on Screen
5 Lower screen area	**SPRITE PROTAGONISTA** Print Screen Background	**DISPAROS** Print Screen Background

With this strategy give just enough time to print every two screen refreshments:

- 1 Hero Sprite 12px*24px a 50fps
- 5 Enemy Sprites up to 16px*16x a 25fps
- 2 Hero Shots 4px*6px a 50fps
- 3 enemy shots 4px*6px a 50fps

It can be seen in the table above that the enemies do not capture the wallpaper, which is very strange. The reason is that the enemies have an area of predictable movements, so we have chosen to capture the background where they move to save time. The buffer where this information is stored is 3Kb. alone, compared to 13Kb, which would be necessary if the whole background was captured.

The protagonist and the shots do capture the background for the same reason:

- because you would need 13Kb. and we don't have them.

- because the movement is not predictable and can be in any area of the screen.

To save more memory space, sprites are stored facing right and rotated as they fly. Fortunately, the design of the enemies by Azicuetano has been symmetrical in movements and it is not necessary to rotate them, but the protagonist could not be symmetrical, so it is necessary to rotate it. The complexity wouldn't be much if it weren't for the interruptions that allow only a few clock cycles to print a sprite of something over 12px*24px at most. If we did the rotation to the flight it would take a lot of times and other times less, reason why the time would not be predictable and we could lose the control and come to have desincronización of the interruptions and to lose the control of absolute way. How was this solved? Very easy. In a conversation with Dr. Fran Gallego he suggested the possibility of keeping the protagonist half rotated and half not. Using this technique we managed to make it take the same amount of time to draw the protagonist looking left or right and also to give time in an interruption.

6.4 Triggers and actions

The game consists of certain adventure components:

- Items that can be collected and placed in inventory.
- Places where inventory items can be used.
- Objects and positions capable of triggering monologues of the protagonist to offer clues and put the player in the story.

All this management is done through a type of special objects where the collision of the protagonist with them is able to trigger the actions commented.

These actions obey a very concrete logic, and a very specific work in programming, since it is not possible to develop a general method to solve such special actions without burdening the performance.

6.5 Compression system

One of the ideas we are especially proud of is the system used to include the intro within the 64kb of the game.

Once the game was finished, with just 150 free bytes, we devised a system to compress the entire game with Exomizer and thus have space to develop an introduction to the game. This intro will only be possible to visualize it as soon as the game is loaded because, once the intro is presented, the game decompresses over all the memory, overwriting the compressed game (with the appropriate safety margin) and the intro.

This implementation is not as simple as it may seem. It has taken some effort in tests and has forced to develop a specific loader, but the benefit of this technique is obvious, and thanks to it we get our desired intro without extra loads.

7. HERRAMIENTAS UTILIZADAS

This project has been developed under Windows and MacOS OS simultaneously.
The main tools used for the development of the project have been:

- Proprietary developed in Python:
    - bitmap2CPC - Sprites.py: convert any bitmap into raw CPC format (sorted by scanlines or in 8 sections)
    - bitmap2CPC - Tiles.py: convert any bitmap to raw CPC format (computer per scanlines or disordered for optimization)
    - screen2CPC.py: converts TMX into raw CPC format usable by the 4MHz.es engine.
- Stiles2cpc.py: converts XML with STILES to ASM format
- Disk generation: ManageDSK, CPCDiskXP and WinAPE 2.0b2
- Generation of CDTs: DINAMIC and CSW2CDT tools
- CPCtelera 1.4 - Prototyping of effects.
- WinAPE 2.0b2 & RMV2 beta - Amstrad Emulation.
- Sublime Text - Code editor.
- Aseprite - Creation of sprites, animations and loading screen.
- Pro-Motion - Creation and management of tiles and supertiles.
- PaintBrush - Quick adjustments in assets
- Arkos tracker - Creation of Music and FX.
- Tilemap - creation of levels.
- Telegram and telephone - Communication between the team.
- Trello - Organization of tasks.
- Documentation: Manual Zilog and http://clrhome.org/table/
- Google Drive and Dropbox - Storage of materials, fonts and assets.
- Google Docs - Generation of documentation.

8. CREDITS

Code & Engine	JAVIER GARCÍA NAVARRO
Gfx & Game Design	RAFA CASTILLO
Music & Fx	JOHN MCKLAIN
Testing	Blackmores
	Metr81
	JGonza
	Pomez666