Making of

1. **The idea**
2. **Software used**
3. **Assembler + CPCTelera**
4. **Lessons learnt**
5. **First stages of the development**

Emilio José Pérez Mariscal
Jesús Cuadra Téllez
Francisco Javier García Fernández

Nibble Games

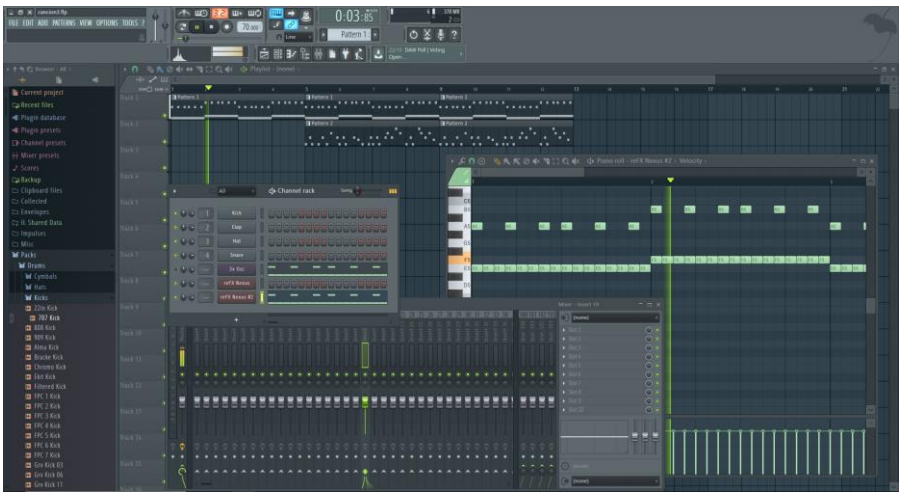
1. The idea

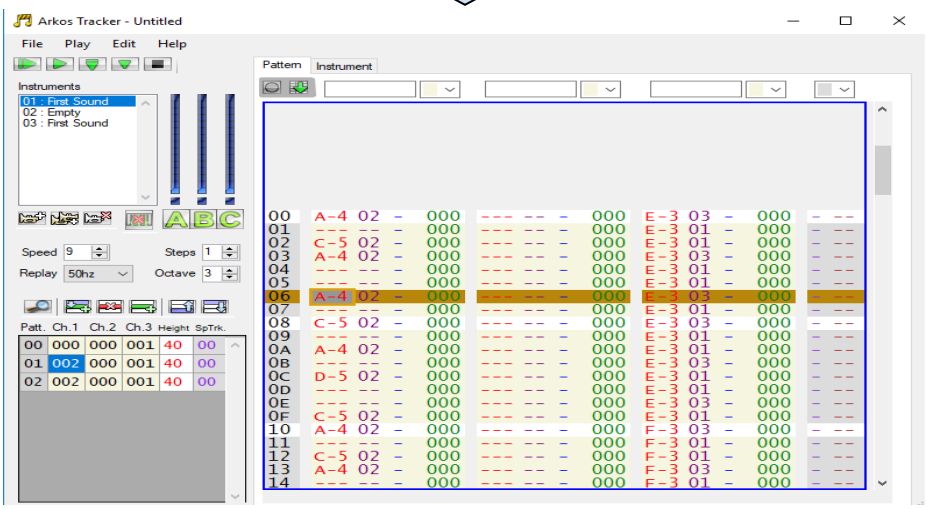
Our first idea was to make a ritmic musical game, but sadly it was too difficult in Amstrad basically it's hard to make a good timing with the music.

We begun this project looking at previous CPCRetroDev games presented in other years. Looking we found a loot of good choices for a nice game. Finally we decided to make a tank fighting game, we thought it was a good idea because it was simple and cool.

2. Software used

Music





- **Software**

At the beginning we used flstudio to create the music because this program has got a pianola editor and it is easier to compose. Then we used Arkos Tracker 1, a software music development in .aks format. Using CPCTelera's music_conversion config file, we added our game music based in our flstudio first creations.

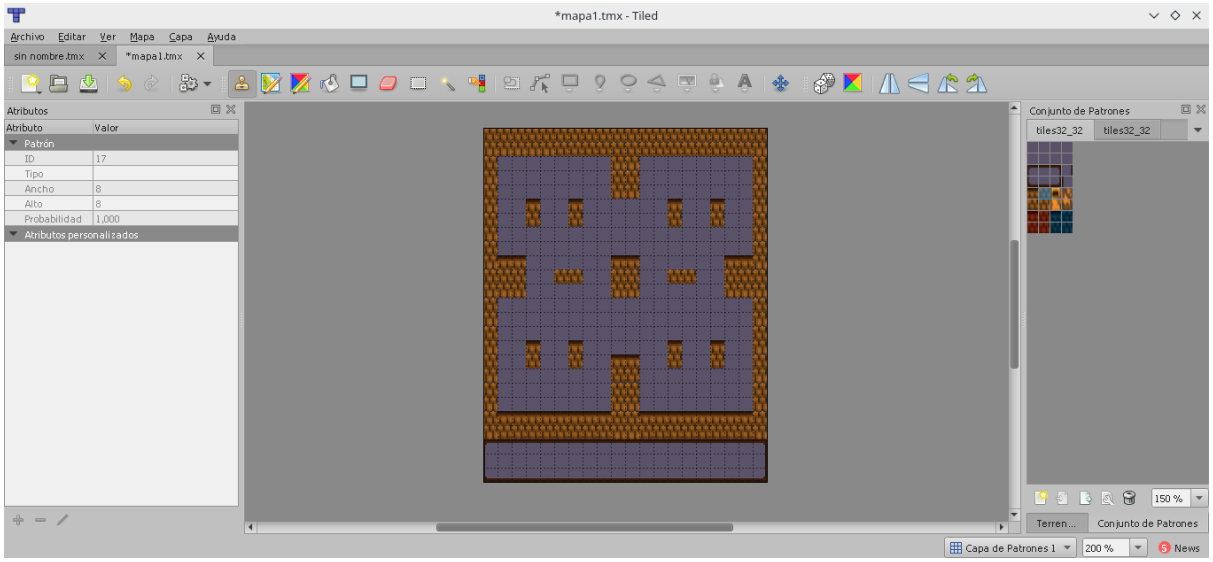
The game has got 4 different songs that are changed between the stages.

- **Problems**

Firstly we installed Arkos Tracker 2.0 but that version wasn't working in our CPCTelera version. At the end, after some test, we discovered that Arkos Tracker 1.0 works perfectly so we used that version. Arkos Tracker 1.0 was difficult to learn, because it hadn't a pianola editor and it was a composing way we weren't used to.

Tiled

- **Software**



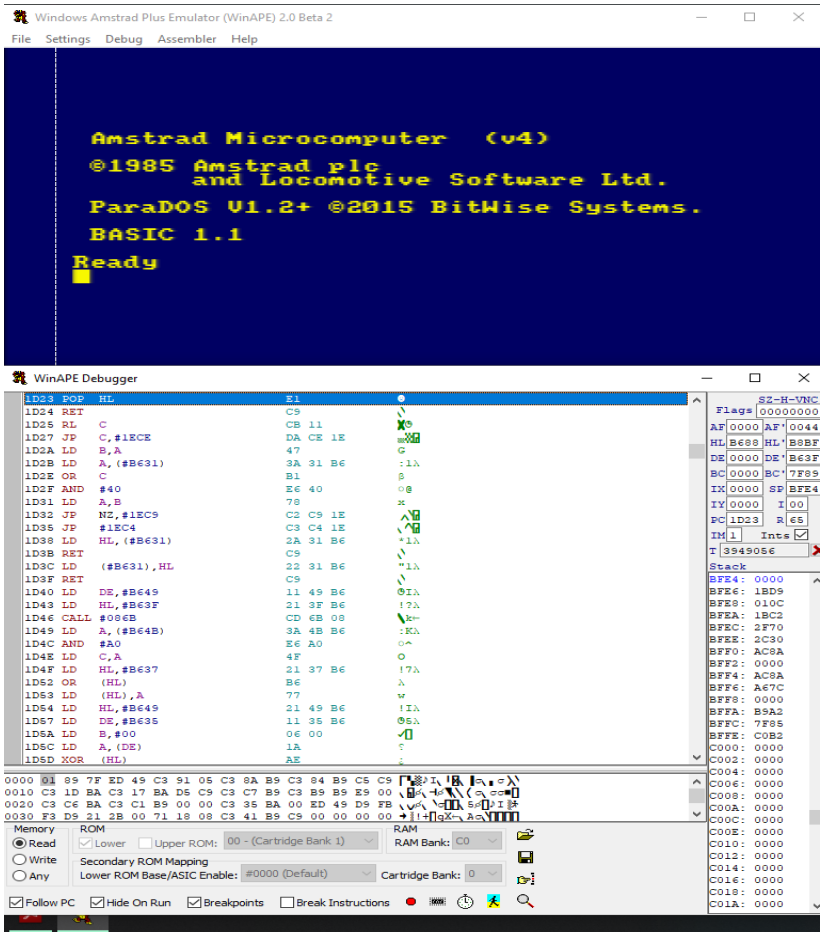
Tiled is a design program that allows to create TileMaps using tilesets

It uses tilesets we created in photoshop to make a stage, using this method we made a lot of maps for our game.

- **Problems**

This program caused us several problems due many factors. First, the size of the map, our primary map had a large width and the map was wrongly printed in the game. Then, the height of the map was the half of the desired, because Amstrad prints half tile instead of a complete one (in mode 0), so we had to create the map with new doubled height size. Next the tilemaps were saving wrongly because we used two tilesets or more incorrectly.

WinApe



- **Software**

Winape is an Amstrad Emulator that allowed us test our game and debug it. The program was very useful because we could use cdt, dsk and snapshot files. Also, the debugging system was a decisive factor because thanks to that we fixed most of our game bugs and inconveniences. WinApe has some helpful tools like recording Session that lets us record a full session and play it back to see the exactly point where a bug was spotted and fix it and others tools like breakpoints. Furthermore we tried RVM sometimes when we hadn't to debug the game because it was faster and it was useful for testing our loading screen.

Using RVM to load cassette file and test our loading screen

- **Problems**

In fact, the only problem we had with this program was the learning of the Amstrad's architecture, at the beginning it was very difficult for us know how to use the registers, breakpoints, debugging…etc. But once we the knowledge necessary, we started working with WinApe without any inconvenience in a fluid and efficient way.

It should be pointed out that we had a problem with our Amstrad profile when we tried to print strings in the screen since if we open WinApe two times in a row, the room data will be corrupted and the strings won't print correctly. Once that problem was located and fixed, we could print every string we wanted in our game.

3. Assembler + CPCTelera

This was the first project for the whole team, where we were programming with assembler.

```
.include "man/st_game_over.h.s"
.include "man/st_ingame_state.h.s"
.include "man_create_nivel.h.s"
.include "man/plant_manager.h.s"
.include "man/shot_manager.h.s"

_gamemode:: .db #05

;; 1 -- 1 vs 3 ia
;; 2 -- 2 vs 2 ia
;; 3 -- 3 vs 1 ia
;; 4 -- 4 contra todos
;; 5 -- Single player



_pausa:: .db #00
;; 0 --> No pausa
;; 1 --> Pausa

_state:: .db #00
;;ESTADOS;;
;;0-pulsa enter para jugar, PRESS TO PLAY;;
;;1-jugando;;
;;2-Game over;;


manage_game::

  ;;Comprobamos en que estado nos hallamos

      ld a, (_state)
      or a
      jp z ,_zerostate    ;;Estado de pulsar enter para jugar
      dec a
      jp z ,_jugando ;;Estado jugando
      dec a
      jp z ,_gameover  ;;Estado Game over



ret


;;Funciones para poner el estado en uno en concreto
set_Inicio::

    ld hl, #_state
    ld (hl), #00
    ld hl, #_gamemode
    ld (hl), #05
    call zerostate_init
```

This was incredible shocking for us, because the ideas, the concepts, the strategies for programing, checking collisions using pointers with registers was crazy for us. We would never imagined we could have used just one Byte to make collisions setting a state for each bit.

Luckily, we learnt fast, we understood how to use registers, make use of some strategies and avoid problems like calling a function that destroys our registers that we were using. Furthermore, he had the help of CPCTelera, that had a lot of functions very useful for us, like drawing Sprites, printing maps or cleaning the screen.

Once we had been programming in assembler for one or two weeks, and we did useful stuff in our game, we had only little and specific drawbacks solved fast and easily.

4. Lessons learnt

- **The use of resources**

We started this project thinking that we had all the resources we wish, and the computer of our game was going to be a incredible powerful machine that would run everything. if this wasn't that way, it was a old computer and it wasn't important.

What a lie.

We learnt, our game could be played everywhere, in any computer, better or worser. We had the habit of doing things like repainting the entire screen or redrawing every element in each frame of the game. When we started doing stuff like this in amstrad, we found it was crazy. Just for redrawing the Tilemap in each update, our game FPS were severely reduced to the point to have an unplayable game. This way we decided to make a game where each element, computation or functionality was used only when it was necessary, reducing the use of our computer efficiency to the minimum possible.

- **A chaos in our code**

At the beginning of the project, everything was going well, we followed up our tasks, the game development progressed and the whole team were working. Until we had a lot of code. When the code was growing up, getting more bytes and bytes of information, more files and more lines, it began to be a chaos.

The solution was doing completely independent systems, for example, we had a music system that worked just with one function call: "next_song". This was very useful because at a point of the game, if we wanted to change the music we just had to call this function. After this, we continued programming that way, which was more efficient.

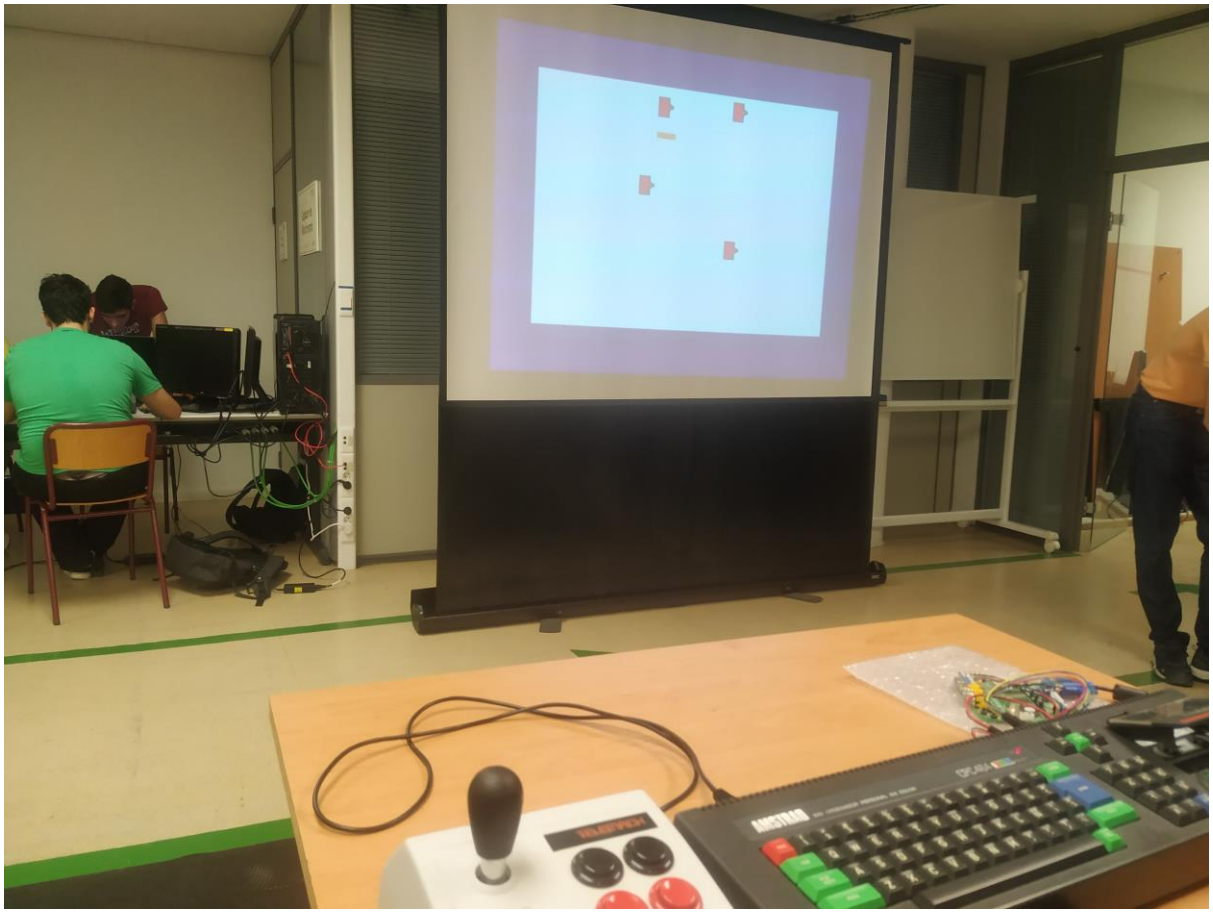
In fact this was not a very huge problem due we realised this inconvenient early, but we had not enough time to make a clean code with good comments.

- **Not enough memory**

As soon as we had our first map in our game, we started developing a system to start the stage setting the enemies' parameters. This ended up being a huge problem because when we wanted to add new maps, it would completely destroy our start stage system. At this point we hadn't time to remake everything we did, which took a lot of time. This way we couldn't do a efficiently map creation system, and our maps were entirely added to our limited 64k memory. If we had created the compression system before, i would have been much better and scalable.

6. First stages of the development

At first, our game had the most básic stuff possible, 4 sprites which were moving around the screen, following some basic patterns or moving by keys pressed. Also those sprites were shooting when pressing space.



Once we had a minimum playable game, we needed a menu and a game over. This way we created our first menu prototipe.

However the text in amstrad mode 0 were too large, and the image were really shabby at first. It caused problems making our hud, so we decided to make our own font for the hud and print it at the bottom of the map.

Finally we created new sprites and set a new map with a nice tileset. This united to our hud made a nice visual game.