MAKING OF
MISS INPUT

Who are we?

We are ChupiGames a group formed by:

- **Daniel Saura Martínez:** Developer of Harvest Day for Amstrad CPC and a member of the group Pixel Productions in 2018.
- **Enrique Vidal Cayuela:** Developer of Crimson Knight Adventures for Amstrad CPC and a member of the group Nibble Games in 2018.

Do you want to contact us?
chupifluxigames@gmail.com(@GamesChupi)

Daniel: danieemil@hotmail.com
Enrique: quiquesoyyo@gmail.com

Technologies used

- CPCtelera 1.5.0: It's a fast Amstrad CPC game engine owned by ronaldo that we used to create our game on Amstrad CPC.
- Tiled: Program used as a tool to create the maps.
- Visual Studio Code: It's a code editing tool that allowed us to write the code of our game.
- Github: Used as storage and as a version control of the game throughout the development.
- Arkos Tracker 1: This program allows us to make our music and sound effects compatible with Amstrad CPC.
- Krita: This program was used to make all the game graphics in pixel art.
- Winape: Amstrad CPC emulator in which we can test our game.

What is Miss Input, cannot detect the keyboard or something?

Miss Input is a platform/adventure game that we developed for the CPCRetroDev 2019 contest. This project is fully made in assembly language and it’s developed for the Amstrad CPC machine, with the help of CPCtelera.

How did you make this awesome game?

Well then, to explain this we'll follow a series of steps ordered chronologically each one explaining our reasoning at every moment and what we did about it.

First step: The Concept

All started with just one phrase: "What if we make a game?".

At first, there were a lot of questions in our empty little heads, "What kind of game do we want?", "How are we gonna do it?", "When are we gonna do it?" and so on.

So we started putting all the ideas that crossed our minds, once we had all the ideas on the document, we chose the idea that we thought was better.

"It's going to be a platform/adventure, (just like super meat boy)."

The next was defining a GDD(Game Design Document), and describing completely the chosen idea. By this we conclude our principal objectives, a good control over the movement of the character and a fluid gameplay.

Also we started defining our player mechanics like jump, movement, wall jump; the different power-ups and the enemies that would be in our game.

In this document was all we wanted to put in our game. Unfortunately, getting all of this into the game wasn't going to be that easy.

Second step: C or assembler?

After the GDD was complete, we first tried making the basics of our game on C… and we failed. Then we decided that we'd create the game in assembly language with the help of CPCtelera, like last year [CPCRetroDev 2018(Harvest Day/Crimson Knight)].

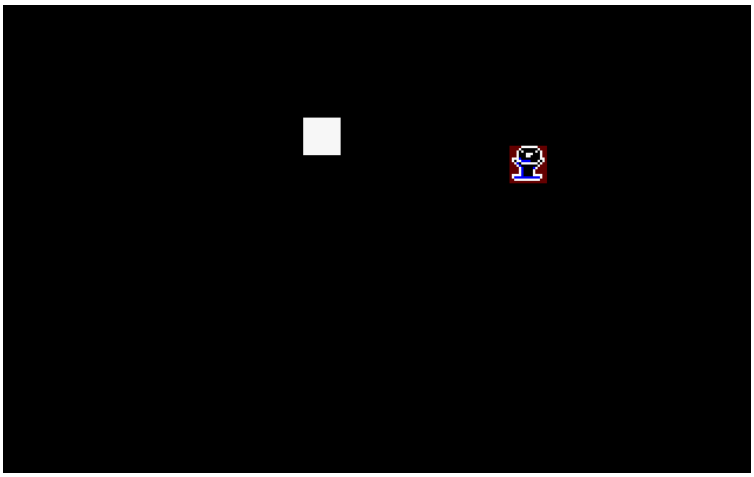
This step started on the 30th of September and lasted until 3rd of October when we changed to do it all in assembly language.

Third step: The Beginnings.

From 3rd to 9th of October.

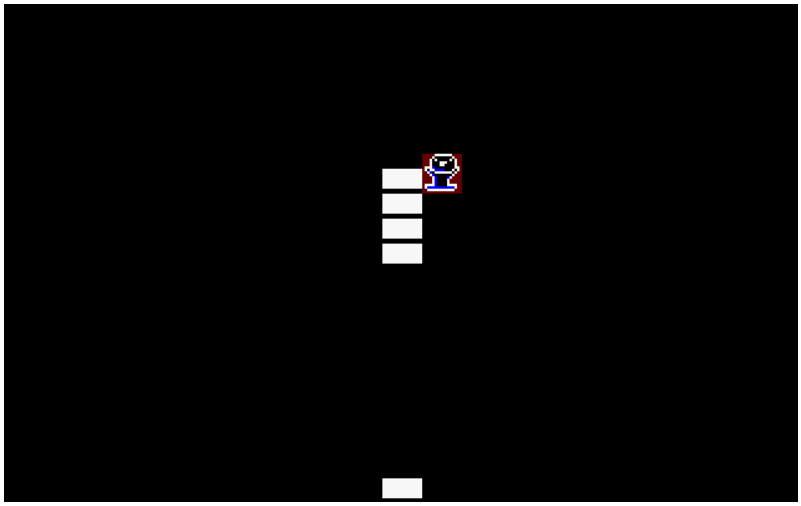
On this stage, we started with the re-learning of CPCtelera 1.5, drawing sprites, cleaning them and drawing the background, how to draw tiles… In summary, the basics to do something that's barely playable.

Also we defined the structure of how it was going to be the player in code and with that we did a basic player with a basic movement, not as we wanted "yet".



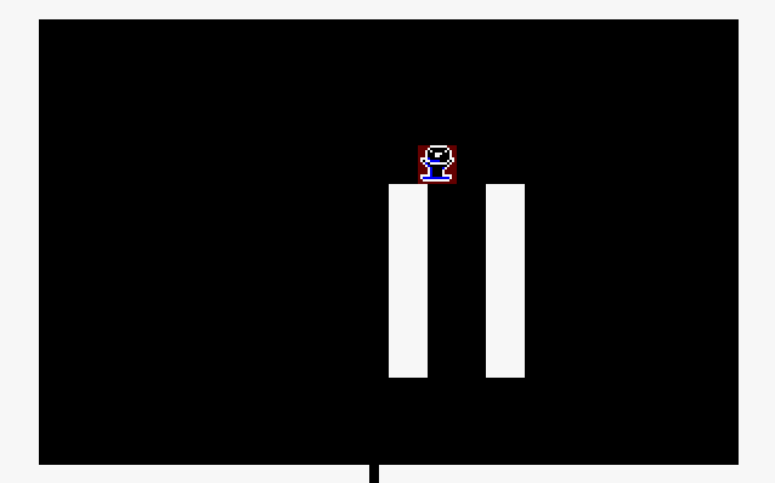
Fourth step: Collisions.

If we wanted our game to be a platform game we'd need a system of collision to be able to collide with the different entities we'd wanted in our game. So we spent an entire week just designing, programming and optimizing our own collision system. We ended tired, but with a consistent way to detect and correct collisions with the player on the 16th of October.



Fifth step: Player mechanics

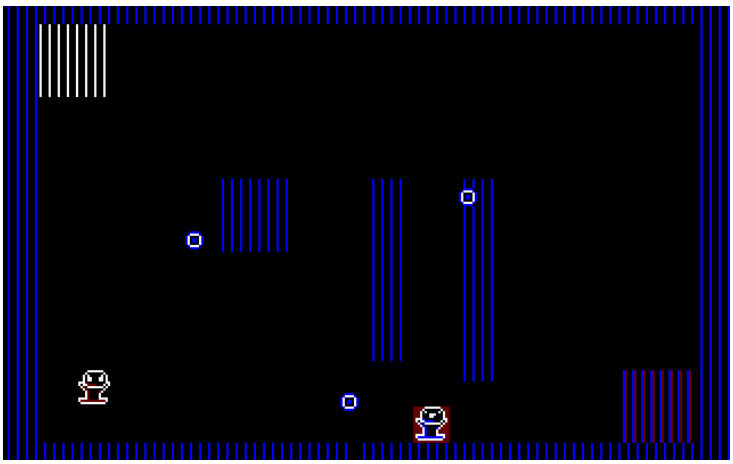
Since the basis of the player was functioning, we began to make the mechanics we wanted the player to get from 17th until 19th of October. These mechanics consisted on making the player be able to jump from the floor and walls, and improve its movement. The jump mechanic was specially centered on a jump table we made that contained all the values of the velocity on the Y axis that would have the player in every moment.



Sixth step: Gravity and power-ups.

The 20th of October we did all the power-ups' structure so that later it was easier to implement the different power-ups. We also made it so that the player could fall up and jump down!(inverted gravity available now).

The power-up of double jump consisted in giving the player the ability to execute the mechanic of jumping once in middle air.
And the other two power-ups were related to the mechanic of gravity previously mentioned, so one power-up made the player fall up, while the other made it fall back down.



Seventh step: Double buffer, redraw and IA.

We spent from 21th to the 23th of October working in parallel, one member improving the drawing mechanics by a double buffer and the another one creating the Artificial Intelligence of the enemies.

Double Buffer:

Due to the speed of the game, we needed to implement the double buffer to avoid flickering. But that came at a cost, a third part of the available programming memory (from 8000 to C000) had to be sacrificed.

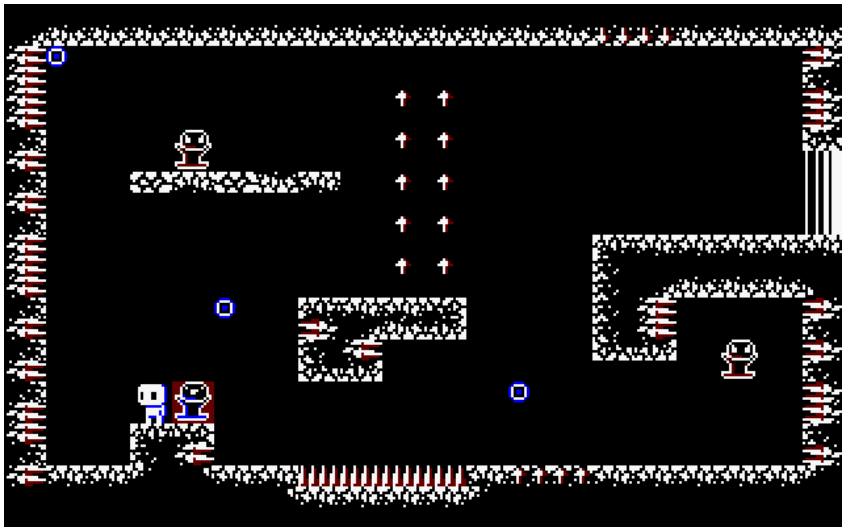
Redraw:

If we wanted a fast game, we needed to optimize the most to ensure high framerate, and that led us to make a function to clean the screen as efficiently as possible. The redraw function only redraws the tilemap section that has been modified by the player or the enemies, all of this having in mind the double buffer, of course.

IA:

We defined before, the behaviour of the different enemies in the GDD, in order to implement them into our code we've classified them in two according to their behaviour:

- The straight enemy: The behaviour of this enemy is basically going in a straight line in a certain direction, although at the same time we can divide this behaviour in two sub behaviours:
    - The bouncing enemy(): When going in a straight line if it reaches the limit predetermined, this enemy will turn back and go in the opposite direction.
    - The resetting enemy(): When going in a straight line if it reaches the limit predetermined, this enemy won't change it's direction, but instead will respawn at its origin position in that level.
- The chasing enemy(Ghost-eye): The behaviour of this enemy is a little more complex than the other two. Initially this enemy will be in a sleeping state, which means that will remain still without chasing you. This enemy has a range of detection that, when the player enters in this range, the enemy will wake up and it'll start chasing the player throughout walls until he dies or rather completes the level.
  Once it has woken up, the enemy will differentiate if the player is near or far adjusting its velocity. Slowing down if the player is near or increasing the speed if it's too far.

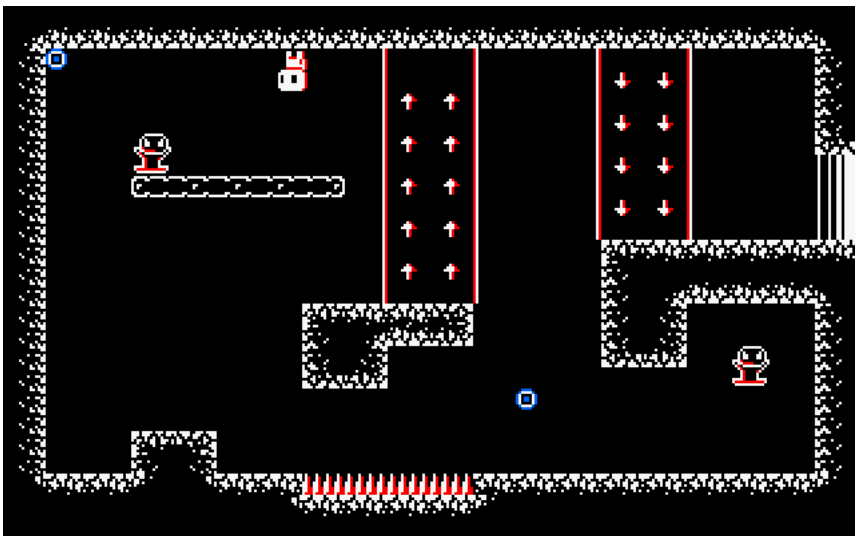


Eighth step: Death and animations

We keep on working in parallel the 24th of October with the objective of improving the player a little more by giving him the capability to die and animate.

In addition to this death mechanic, we programmed the level restart to be able to play the level again once you died.

And with the animations included we could see the character visually interacting with its environment which gave the user more feedback about the actions he was making.

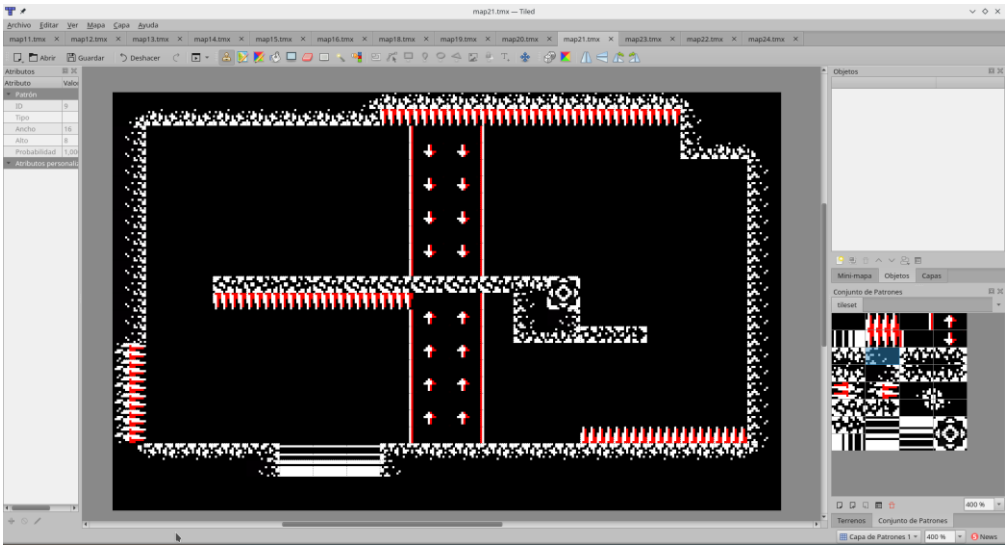


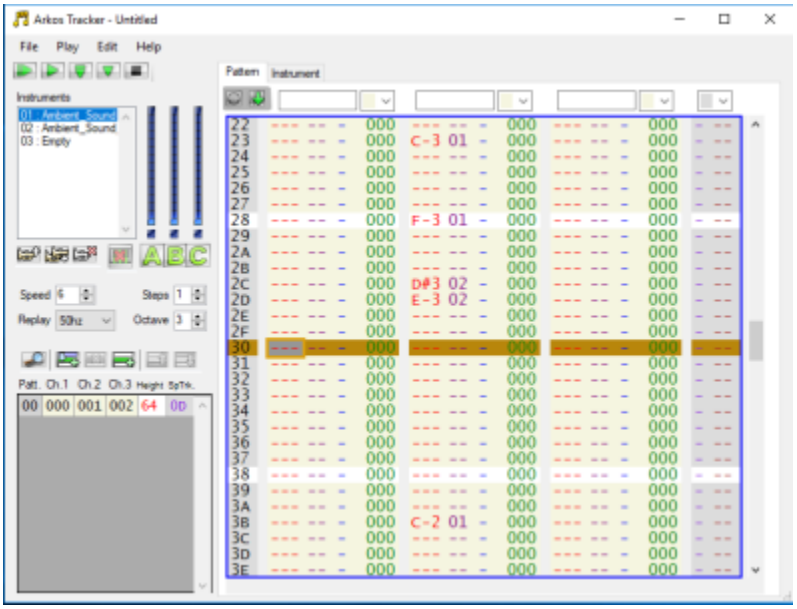
Ninth step: Level factory and music

The days 25th and 26th of October were dedicated to make a level factory and produce music.

The level factory was made to be able to create levels easily and without any problems. In this level factory we splitted up the logical part and the visual part of the levels in two. Therefore to make a level we had to create the visual part on Tiled and export it to our game, after this we had to create each one of the correspondent entities inside our code that matched that map. *So tedious*.

We made the music with arkos tracker 1, the reason for making it in this program is that CPCtelera allowed us to export it and convert it into our game easily. PD: We aren't musicians, don't expect too much from it.





Tenth step: Content creation time has come

On the 27th of October we were fully dedicated to level creation and more music composing.

We even improved the way music was being played in the code by adding interruptions that let us reproduce our music in “parallel” with the rest of our game. With this improvement we also included some sound effects because the music was mostly ambiental. Read the PD on the ninth step if you want to understand why.

Also it included a mode in our level factory so we could include a graphic palette on each level. It’s a shame we didn’t get to use it at all.

Eleventh step: Menus, loader and the reference

The next day, 28th of October, we worked on the front end part, this means we included menus in our game, we also did a loader that let us show our loader screen while the game was still loading and finally we included a reference to Astro Marine Corps. All these upgrades made our game improve as a “product”.

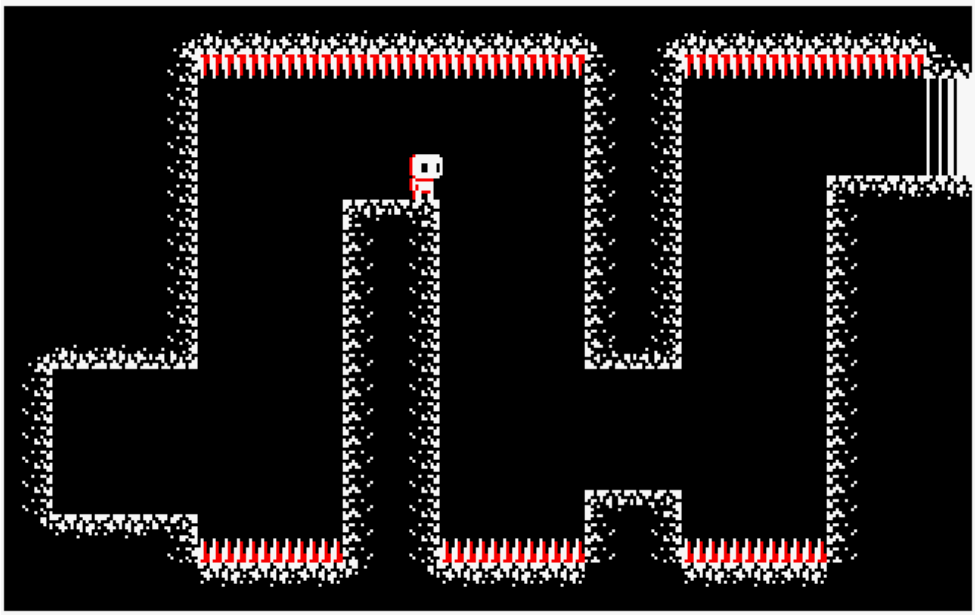
Our menus were basically an entire screen sized image that we put on screen during a certain time or until we pressed a certain key, and then we jumped to another menu image or to the game execution.

The loader was mostly made by CPCtelera so we didn’t need to invest too much time on it. God bless you CPCtelera authors.

Twelfth and final step: More levels, testing and license

This step was the final sprint that lasted from 29th of October till the deadline 30th of October. In this sprint we spent time making more content, a lot of levels. And, of course, testing our game searching for any minimal bug that could lead us to destruction and despair.

The last day was mostly dedicated to obtaining all the licenses, mentioning all the external libraries we've used, gameplays of Miss Input and the redaction of all the documentation needed. Yes, you're right, we made this on the last day and we don't regret it, not even a bit.



Problems found:

The first problem we found occurred when we tried to make our game on C due to the fact that we hadn't made a game before in C language and for Amstrad CPC. Furthermore, there was a lot of knowledge we didn't have, we couldn't work at a low level so easily like in assembly language.

The little time we had once we started was a big problem, we had only 1 month to make entirely our game. Luckily we have some experience now of last year's contest.
Why didn't we start before, whyyyyy!?

At the start when we were using CPCtelera a few times some bugs appeared that weren't related to our code. Surprise!!, they were small bugs that had CPCtelera. Fixing these bugs took us some time we didn't have.

Lessons learned:

The later we start to make a project the less time we'll have to polish the game, if we even finish the game.

If we're using an external tool or library, it'd be useful to understand how it works, so if that fails it can be fixed later on and be easier to work with.

Planification is important if you have a ton of work to do and too little time to do so. Without planification, a lot of time with a few tasks to do can change into a lot of work without enough time to complete it.