We are Pixel Productions and here's all the work that we had to go through, explained. Also we'll explain the technologies used, the problems encountered and all the lessons learned during the creation of the game.

Technologies

This is the list of technologies and what we used these for:

CPCtelera

CPCtelera is the game engine for amstrad made by Francisco Gallego that we've used on the development of our game.

We have used it for the development of the entire game through the functions that offers CPCtelera, that we'll be explaining throughout this document.

Winape

This amstrad emulator has been highly used to test all the features implemented on our game during all the development of our game. In this document all the captures done from the development phases will be taken from the winape screen. We will talk later on the features we have used of this technology to fix the problems encountered.

Arkos tracker 1

Arkos Tracker 1 is a software to build sound effects to computers like Amstrad, Atari or Spectrum. We made the background music using this program.

CPCtelera was used then to convert the aks file into a readable music for the compiler.

Gimp

Gimp is a 2D image editing tool that we used for the creation of all our sprites and tiles of the game. Later on, we will be explaining how we created our sprites and tiles to be able to implement them into our game.

Sublime text 3

Sublime Text is a text and code editor, used for the creation of all the source code of our game with a Z80 Assembly syntax highlighting added via package control.

Github

It allow us to work in parallel and then put it all together minimizing conflicts. We've also used it for the storage of all the project and for the version control.

Tiled

Tiled is a map editor that we used to design the map of our game with a combination of tiles. We also used Tiled to personalize the characters, enemies and objects state in each level.

Development process

To understand how this game was developed we'll explain the progress made on each week during the development of Harvest Day. Furthermore, this game was entirely made on assembly language.

Definition of the game

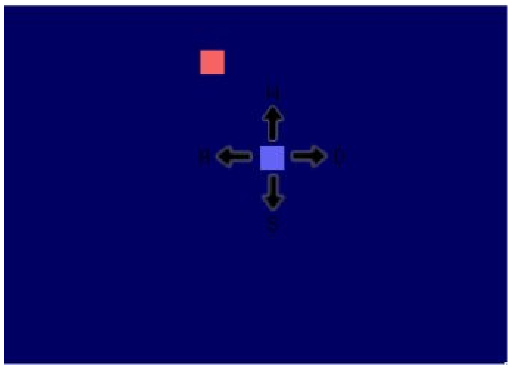
On the first week, we decided the game model on which we were going to base, to have a reference at the time to implement all the mechanics needed.
At the same time, we had to do a research on all the technologies we were going to use for this project, and so, learn how to use them wisely.

Second week of research

On the second week we continued the research of all the technologies, more extended on how we could use CPCtelera to our advantage. We barely implemented some basic mechanics to our game as well:

Basic control of the player:
Detects the inputs via keyboard and moves the player depending the key that it's pressed.

Implementation of an enemy:
An entity "enemy" that only moves in one determined direction.



Third and last week of research

We kept on researching and testing CPCtelera to reassure the knowledge acquired on the first 2 weeks, ending this week we ended the research period. Moreover, we implemented a few more basic mechanics this week that would set the bases of the project:

Double buffer:
This technique was needed to avoid the flickering, however it presented memory and performance problems.

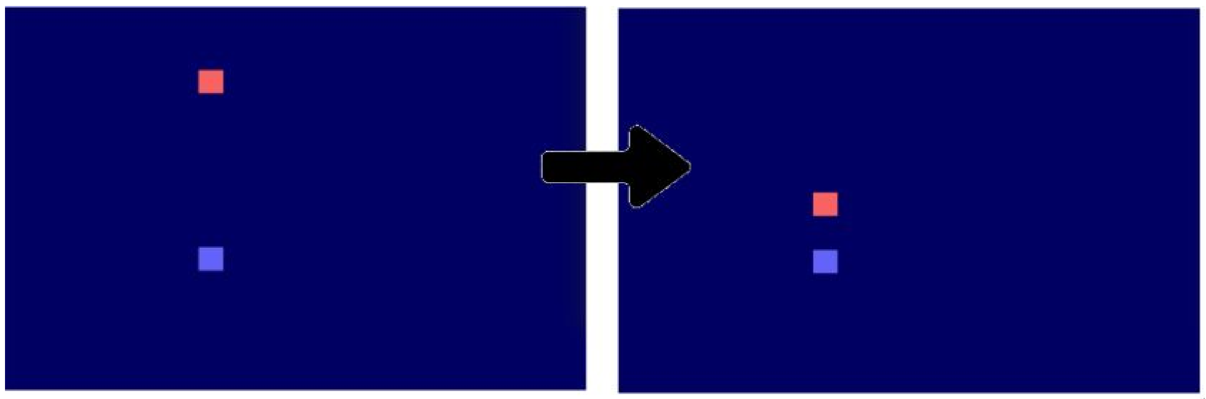
Arrays:
Utilized to manipulate entities of the same type easily and efficiently.

Task assignment

Fourth week, first week of developing our mechanics. Once the research period was over and we had the base to start testing and developing the mechanics we needed, we divided tasks so each member of the team could prove and test the pending mechanics individually, not including them yet into the project:

Matrix implementation:
This allowed us to discretize our map easily, it was used mainly as a collision map, so we could include more entities reducing the cost of checking the collision with every entity to just check if where i’m moving is something that can pick up, block or kill the entity.

Basic enemy AI:
A simple enemy that could follow the player all the time without any check of collisions.

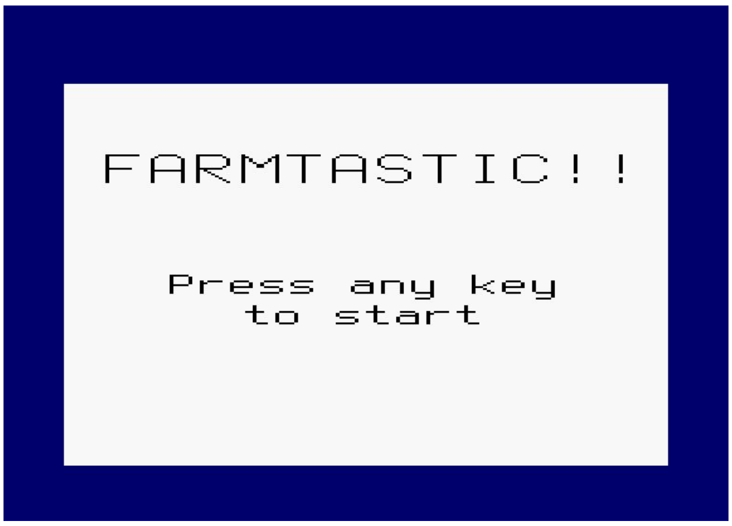


Basic collectables:
A basic static entity so the player could pick it up and the enemies could pass through without affecting it.

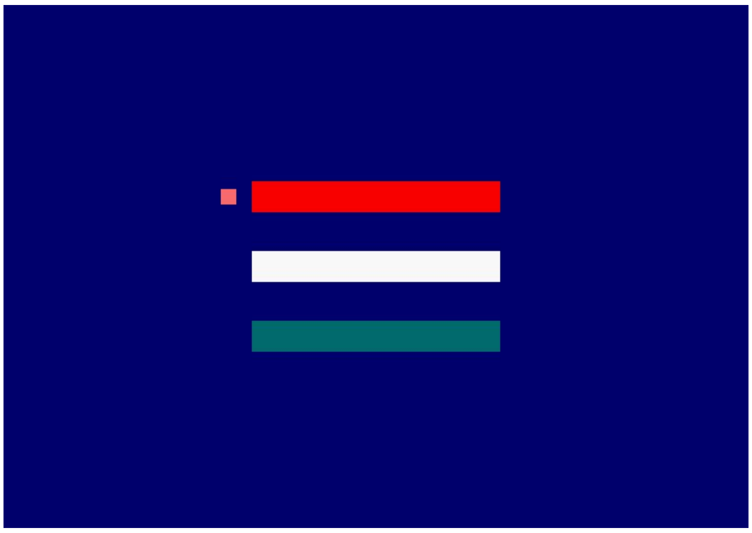
Double buffer improved:
Cleaning and redrawing all the screen was so expensive computationally, so we developed a variation of this technique. This complex variation consisted on cleaning and drawing only on the part of the screen that was changing, the performance improved, not dramatically though. Cleaning and drawing a “tile” in each iteration for each entity that was moving was still expensive.

Menus:
A simple initial screen before getting on the game that allows the user to choose when he wants to start the game finish it, or even pause the game pressing a certain key. This menus were all in black and white.

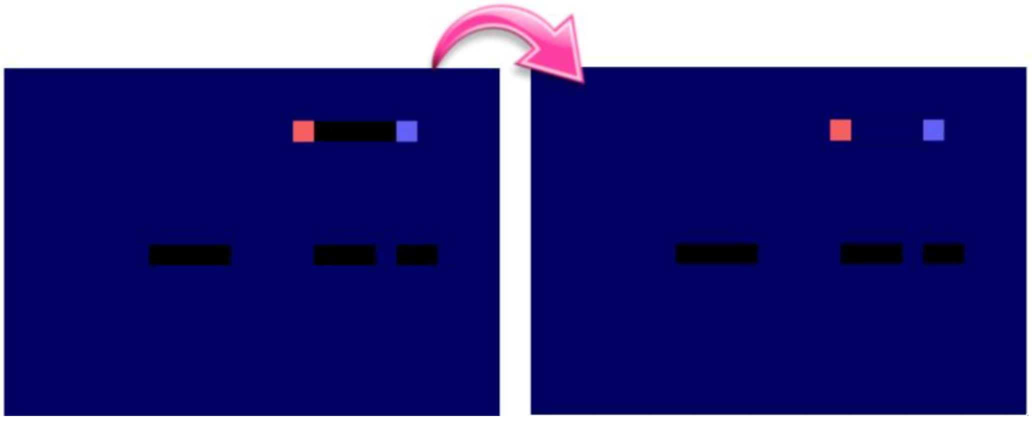
Start menu



Pause menu



Construction and destruction:
This mechanic is that the player could create or destroy a row of obstacles depending on where it was and where it was looking. Also the enemy that chases the player should be able to destroy one obstacle if it was on his way.



Putting it all together

Fifth week, second week of developing our mechanics. We had all this mechanics done all apart from the same base of the third week, so we decided to piece it all together before the code got more complex, and optimize it too. That way we could establish these essential mechanics as a base for further mechanics.

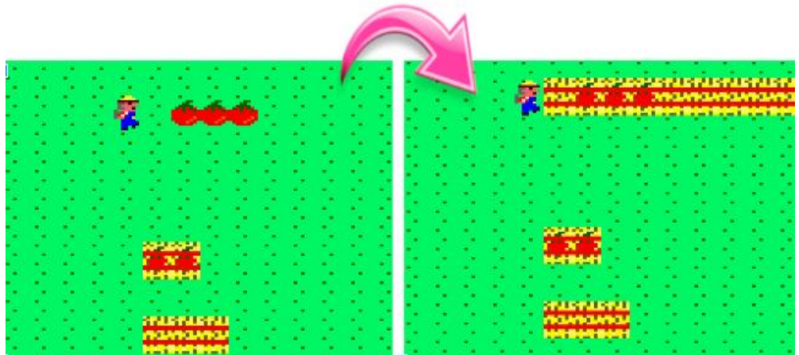
This took a lot of effort and time, we spent all the time of the week and more, implementing all these mechanics from the base one by one optimizing them all we could so we could get a solid base after to be able to work better and more efficient.

New advanced mechanics

Sixth week, final week of developing our mechanics and first week of adding content to our game. On these week, we added some new mechanics that would improve the gameplay, some of this new mechanics were based on the mechanics we already had:

Obstruction of the collectables:

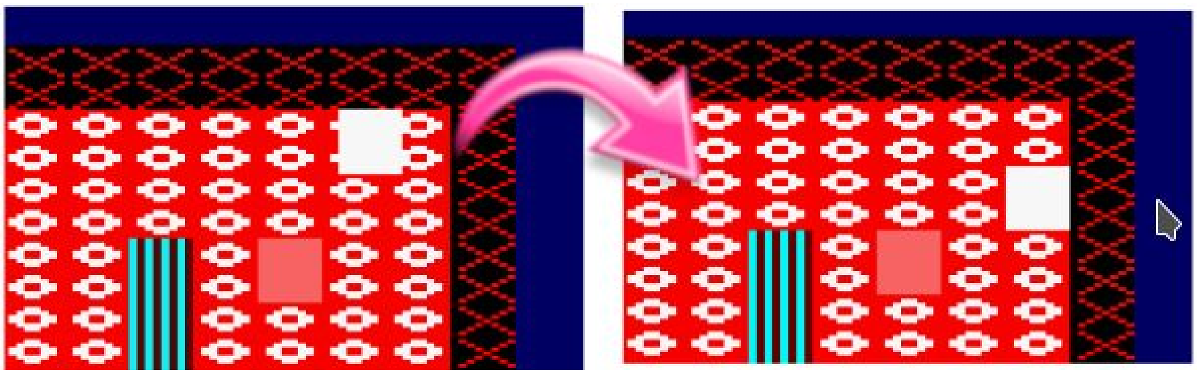
This concept is based on the mechanic of creation and destruction. The case is produced when a player creates a row of



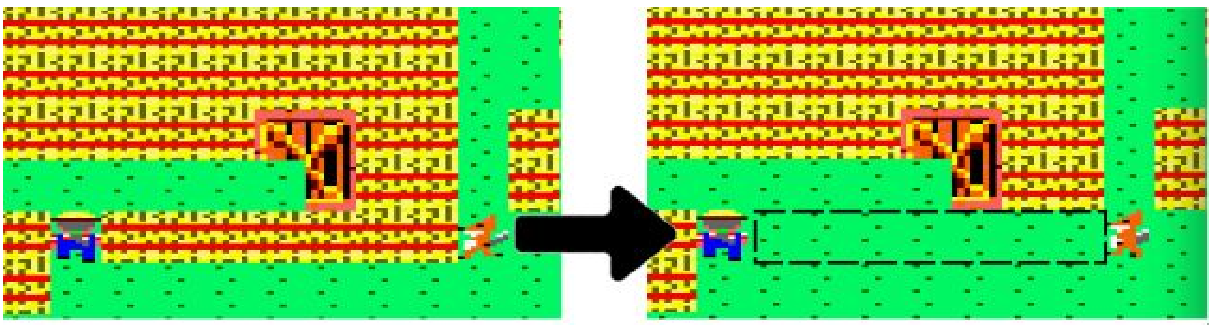
obstacles and a collectable is on the way of one of those obstacles created, in that case the collectable will be trapped in that obstacle until some entity breaks that obstacle and sets it free.

Improve of the enemy AI:
Now the enemies that chase the player, can collision with other enemies and can break their way through the player if there's an obstacle in their way. Also it makes decisions where it has to move or break depending the matrix of the level.

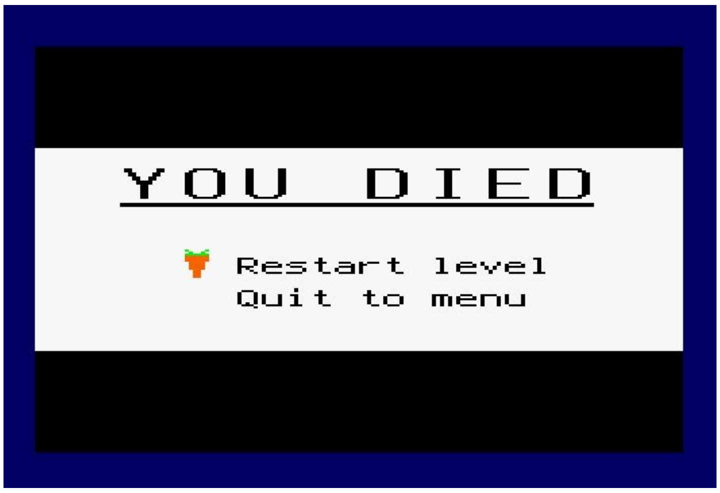
New enemy bouncing:
Simple enemy that turns to the right every time it collisions against a wall, obstacle or enemy.

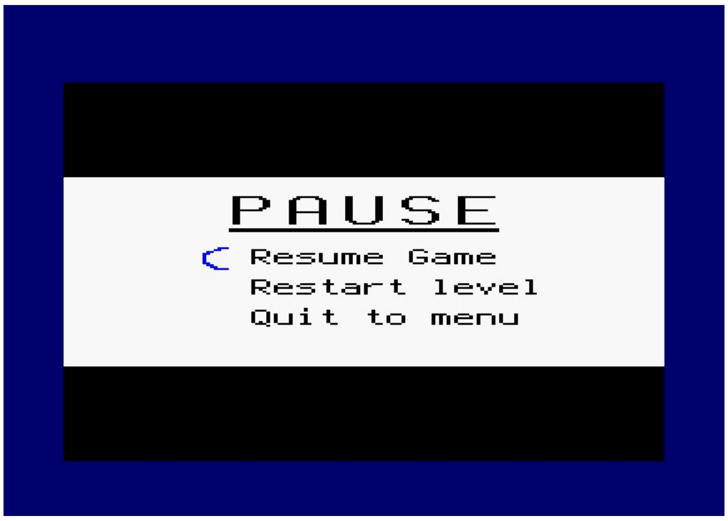


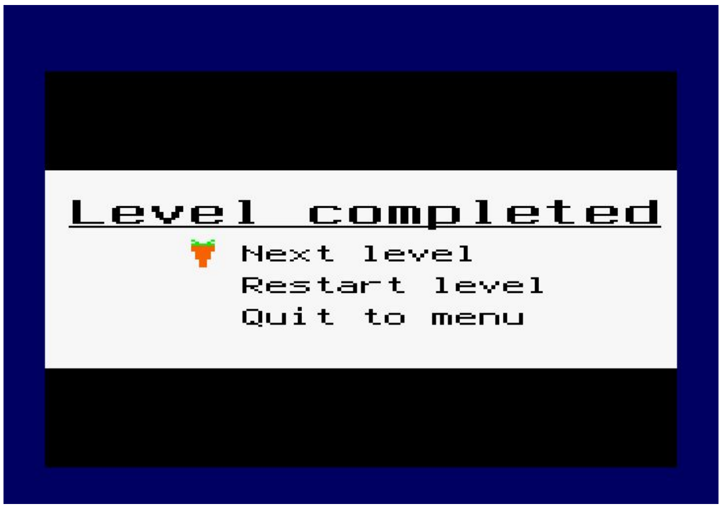
New enemy breaker:
It's completely based on the enemy AI(enemy chase), the only difference is that this type of enemy can break a row of obstacles just like the player does.



More menus:
We've implemented the menu when the player dies and when he completes a level.



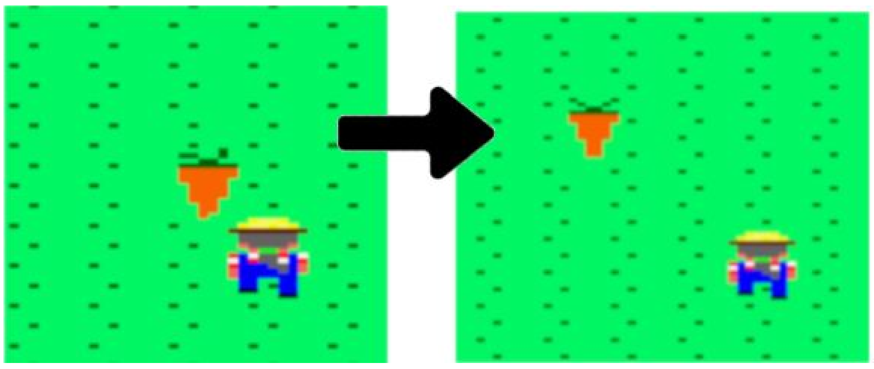




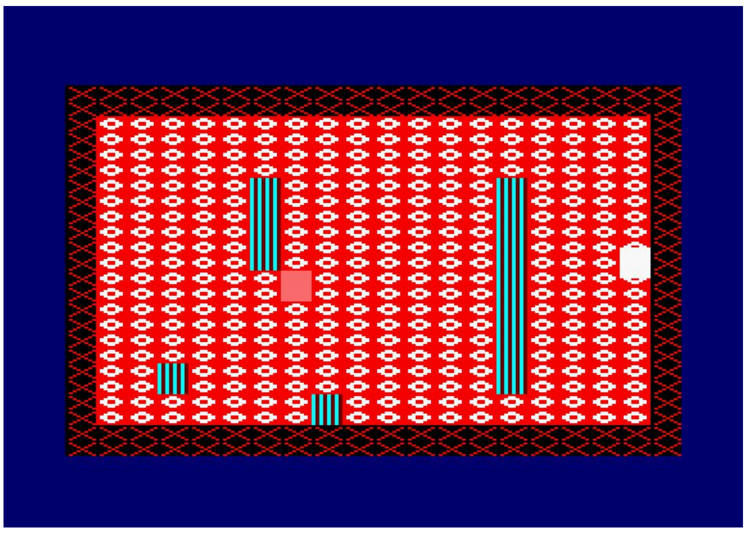
Level charger:
With this mechanic we can pass to our game a level designed on Tiled and this level manager will process all the tiles of that level and will create all the entities defined on it, and will set them in the matrix. This mechanic is used at the start of the first level, when the user restarts the level and when the user passes to the next level.

Collectable with AI:

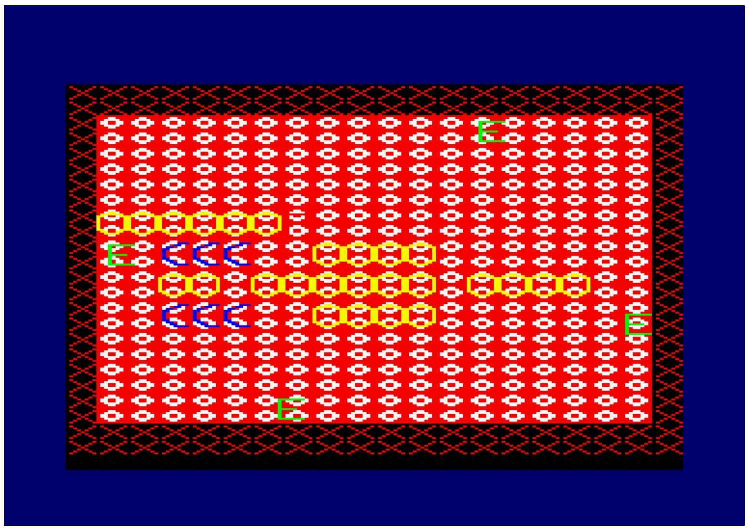
It adds the concept of a collectable with AI that avoids the player at all costs. This behavior is the opposite to the enemy chase.



Tileset and tilemap:
With this we included a background to our game drawing the different types of background with Tiled. These tiles were merely to test other mechanics later on we would add the definitive tiles for the map.



Sprites masked:
At this point of the development we thought of a function that offers CPCtelera where the sprites have a mask, allowing us to set a part of the sprite transparent.



Add content to the game

Seventh week, second week of adding content to our game. In this week we dedicated most of the time to add the content that was supposed to be on the previous week and this week. Also we started correcting errors and bugs that could appear due to the concatenation of all these mechanics together. In this week, we decided the artistic style of our game.

The content created during this week was:

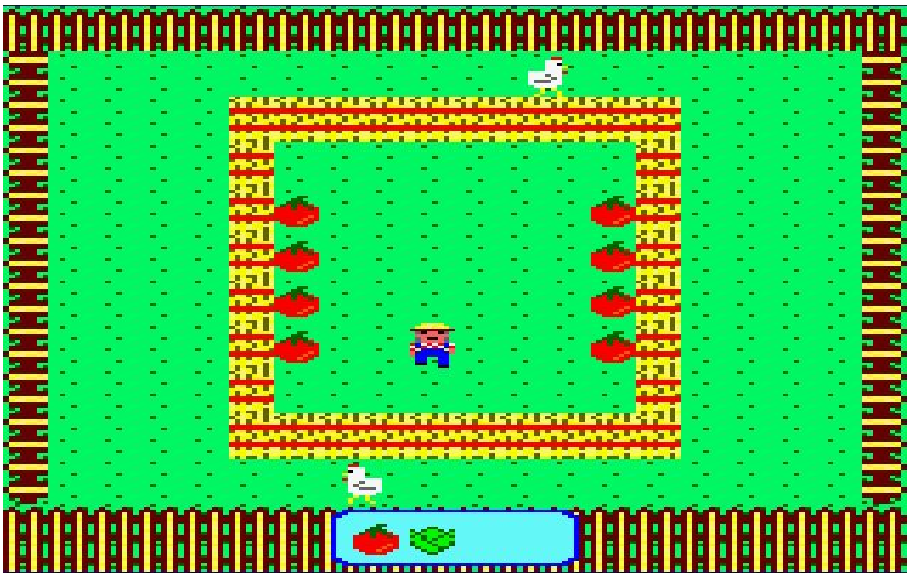
Sprites:
The sprites were created using gimp, with 8x16 pixels of dimension, these proportions were restricted to the size of each section of the matrix, that we defined first. Amstrad’s 0 mode converts the 8x16 into 16x16, doubling the column pixel.
Of all 4 types of collectables, the 3 types of enemies, the player, the HUD, all the menus, the obstacles and the 4 types of collectables trapped.

Animations:
Of the collectable avoid and bouncing, the 3 types of enemies and the player. The animations have 2 sprites, one that is used in the beginning and the end of the movement and other one that is used in the middle.

Levels:
All of the levels are builded to get the most player attention. All the mechanics of the game are shown progressively through the levels.

Music:
Game theme is a “typical day at farm”, so we wanted to transmit that emotion to the player, making a simple but catchy song.

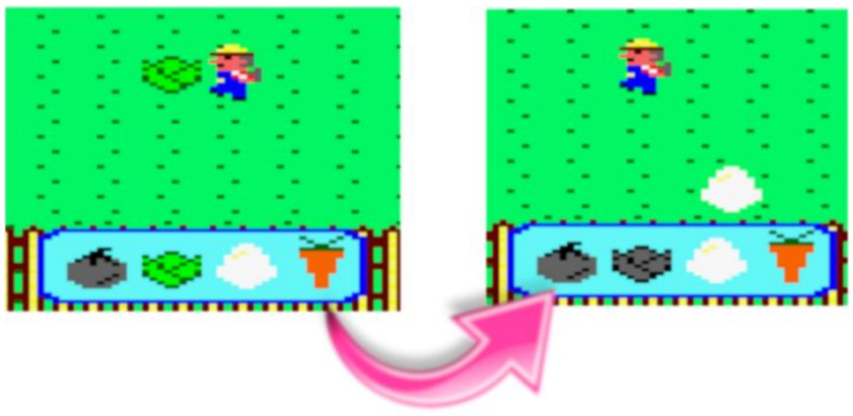
Tileset and tilemap:
We changed the testing tilesets and tilemap for others that matched our graphic.



Not only we created a lot of content this week, we’ve also implemented a few more final mechanics with their respective testing and correcting:

Collectable bouncing:
It’s a collectable whose behavior is the same to the enemy bouncing.

HUD:
Interface located down the screen during the game that shows all the collectables in that level, and turns on gray the type of collectables the player has already picked all up.



Delaying:
Adds some delay to all the entities that move to obtain a movement more realistic of the entities. Moreover, it's essential at the time of implementing the animations.

Remove mask:
We soon realize that this map of transparencies was expensive computationally and we couldn't afford it. So we removed the mask and we planned drawing the sprites with the background on them.

We spend about 2 days of the week to fix some casual errors, and trying to optimize some parts of the code hoping we could gain some performance.

Final changes

Eighth week, third and last week of adding content to our game. In this week, we tried to fix all of the bugs we found in our game, most of them relatives to collisions between entities and movement. We also prepare the differents documents we had to have for the competition, like this Making of, the Manual, some snapshots and a gameplay.

Problems found

Performance:
One of the most important problems we found was the performance limitation of the Amstrad CPC. This supposed a major problem to us since we made all the planification of the game ignoring the computational limitations of Amstrad. The solution we applied into our project was the optimization of great part of our code and some mechanics were removed or simplified.

Memory:
Another problem was the memory limitation to 16k bytes of Amstrad. Moreover, with the implementation of the double buffer, half of the this memory was reserved to video, also all the sprites, levels and music, occupied a lot of the memory that was left. So we compressed the maps, levels and menu's images to be able to add more content to our game.

Flickering:
This problem was produced when we wanted to draw a bunch of entities in every iteration, but we were slower than the raster of the screen, so sometimes an entity wasn't entirely drawn on the screen. This problem was solved thanks to the double buffer technique, that also lowered the performance and the memory we could use.

Sprites:
Using Amstrad's 0 mode which double the column made a little more difficult the sprite's design because we had less pixels to draw and each pixel will be doubled in the game.

Lessons learned

With the realization of this project we learned some of the bases of game design. We practice how to make a game in a short period of time and how to organize the work that has to be done giving priority to some task over others.

Also we learnt the hard way to keep in mind the limitations of the target machine before developing all the mechanics because the most important thing about a game is that it can be played correctly and it isn't how many different mechanics it has.