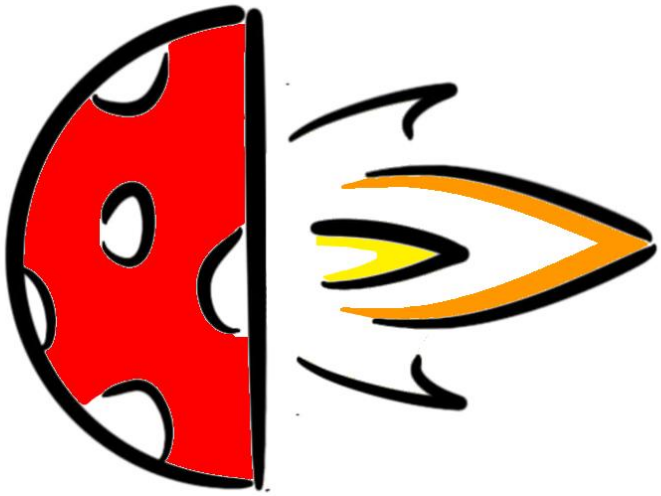
THE LAST FUNGUS



MAKING OF

Index

About us

We are **Ocacho Games,** a game development studio created in July of 2020 as a studio used for game jams and game university projects. We are made up of 6 students, but for this project, as the **CPCRetrodev** says, we are only 3:

-
- Antonio José Fernández Belliure
- Miguel Pérez Tarruella
- David Martínez García

OcachoGames's Twitter

About "The Last Fungus"

We wanted an hectic and addictive game. For this reason, we created *The Last Fungus,* a fighting game where you should defeat a bunch of enemies in order to claim your woods and protect your people.

We also love multiplayer games. The game contains a local-multiplayer mode where you can fight against your friends to see who is the best!

The Last Fungus is the game we developed for the **CPCRetroDev 2021.**

The game is entirely developed in assembler code, using the Amstrad game engine ***CPCtelera*.**

In case you are interested, you can also run this game on a real **Amstrad CPC 464 machine**.

TECHNOLOGIES USED DURING THE DEVELOPMENT OF THE GAME

- Visual Code → Text editor
- CPCTelera → Amstrad game engine
- Gimp → Image editor to create the art of the game
- Pixelorama → Pixel art tool
- Arkos Tracker → Musical program used for producing music

Early stages of development

Amstrad was a completely new machine for us. The first weeks of development were about learning what Amstrad is, how the Z80 assembler works and the limitations that we had.

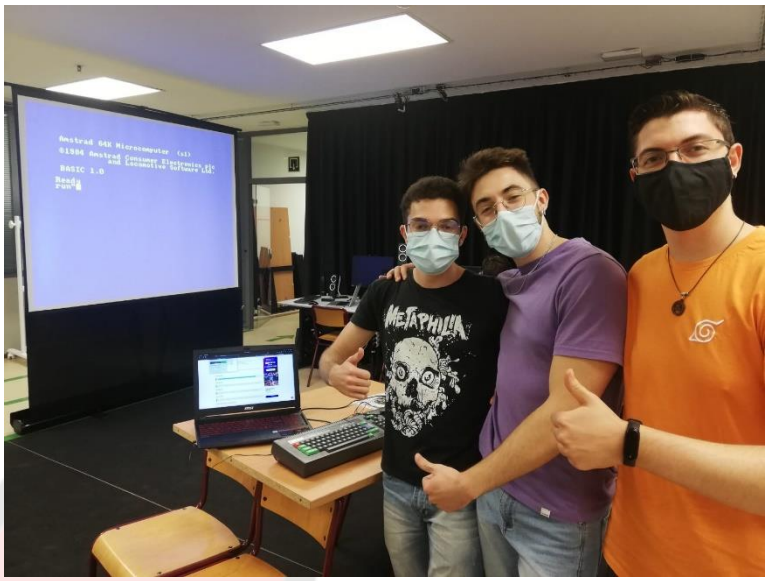


IMAGE 1: DEVELOPMENT TEAM WORKING WITH A REAL AMSTRAD CPC 464

Going deep on assembler code was an incredible experience for us. Working on low level programming (assembler) gives us the knowledge to optimize our code. From our perspective, every good programmer should give a try to assembler code in order to improve their skills and the memory management of the machine.

THE IDEA

Our idea was crystal clear from the beginning. We wanted a hectic fighting game with single and multiplayer mode. Naturally, our game idea was getting better along the development of the project with the ideas and help of the people. We have multiplayer mode and over 6 controls.

For that reason, we put a lot of effort into the game feeling of the controls. The feedback of the people was our best weapon.

At the beginning, as any game begins, we only had cubes and the most basic mechanic, move.

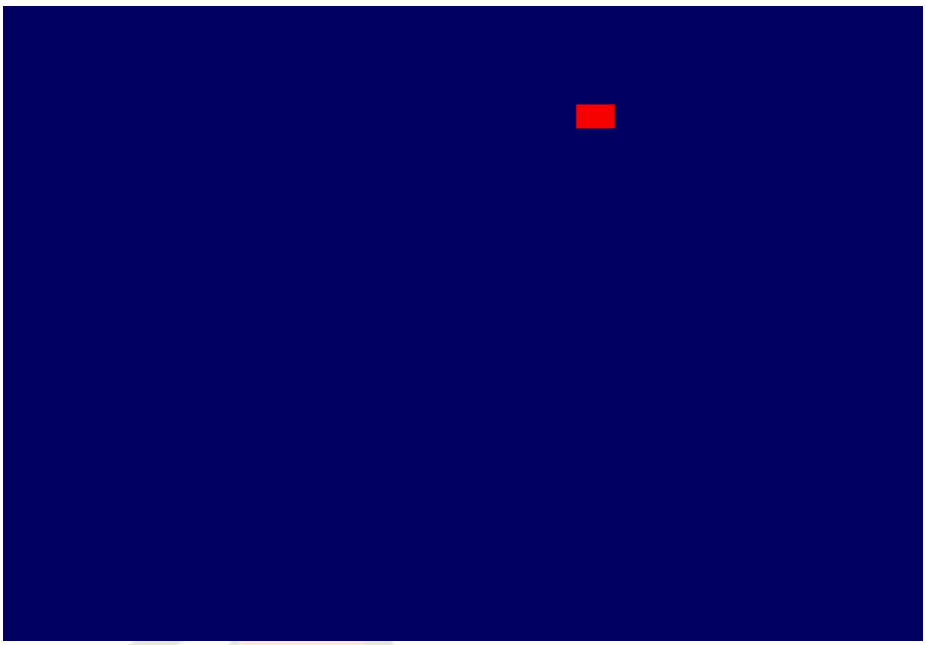


IMAGE 2: EARLY STAGE OF THE MOVEMENT.

Once we had the movement, the next step was shooting. Shooting was a little tricky because of the multiplayer. Each player has their own bullet reference in order to know if they can shoot or not, once the bullet is destroyed, they can shoot again.

Shooting bans you from moving, which is the key of our game. This idea didn't come in the first iterations of the game. It was after playing and with a bit of feedback that we arrived at this conclusion.

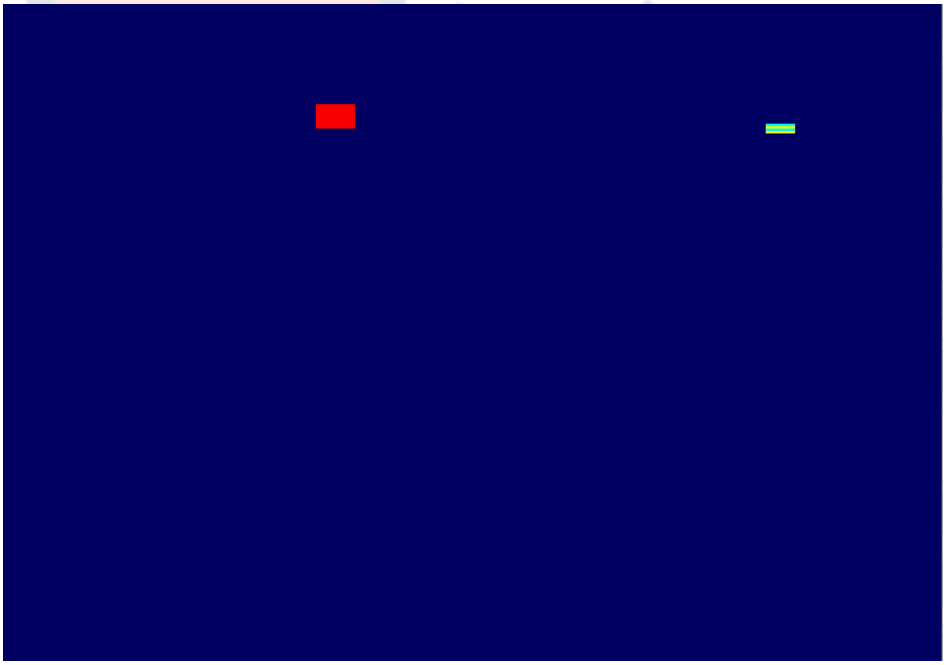


IMAGE 3: EARLY STAGEOF THE SHOOTING MODE

In the shooting mode, you control the bullet in order to stop it to wait until your enemy is on the right point or to boost it to reach your target faster. This mechanic was one of our first ideas.

To be honest, increasing and decreasing the bullet's velocity was our first idea, but it was a little bit rusty to play with. Otherwise, boosting and stopping, it was a lot easier to play and have fun, which we really liked.

As a multiplayer game, we needed to clone the behaviour of our main player to the second one. Choosing the controls for both players was hard, but we overcame the problem by putting the hands of the players as far as we can in order to get a better game feeling while playing with a friend. Also, joysticks are available for playing, that's a good point for multiplayer mode.

To finish this chapter, we would like to talk about the early collisions of The Last Fungus.

Our goal on these stages was to test the game. We wanted a simple collision to check if our game was working correctly.

So we did it, checking the border of the screen to ban the player from getting out of it. On the other hand, the collisions bullet-platform was checking the X axis of both entities.

Middle stages of development

Once we had the main mechanics of our game it was time to divide us and start working on different things. We made three main parts, the gameplay, which included everything related with the input system, the movement and the action of the game; the AI, which firstly included only one type of AI (enemy) and we wanted the behaviour to be like another player and the third part was the render.

GAMEPLAY

For the gameplay, now that we could move the player and shoot we continue implementing the collision for the bullets against the players, so we could kill the other player. Firstly the bullets were going to have collisions only against the players, but as we were progressing with the game we thought that it would be funnier if the bullets could have collision ones against others so the player had one more way to avoid the enemy bullet.

When we had a system to shoot and the bullet had collisions we implemented the system to make the players die when a bullet hits them, so we did and once we had it we only needed to implement the jump to have all the mechanics for our game. We implemented a gravity system and a jump system that worked with fixed positions in the Y axis because we didn't have collisions against the map.

That's the last thing we implemented for the gameplay in this phase of the development, the collisions against the map, which first we made a tilemap and implemented the collisions against the tiles. Later we decided to not make tilemaps so we had to remade the collision system, and we made the platforms of the map to be entities so we could check their positions in each map to calculate the collisions between the players and the platforms of our game. Once we had finished with that we thought that it would be funny that the bullets collided with the platforms too, and so we did.

Later we made some proves with the game by letting some people play it, and we received some feedback such as making the player to not move when he has shot so it has a little handicap to shoot and should think twice before shooting.

AI

For the AI, as commented before, we first thought to make only one type of enemy. To achieve this we started by making a behaviour system for the AI, which included a patrol system to establish how the enemy moves. Once we had this we should implement a system for the AI to shoot at the player and to avoid some of the bullets of the player.

We made this first part and once we had it working, with a lot of things to polish, we realized that it could be a little difficult to implement the jump system for the AI to be as good as we wanted for the game experience, so we decided to rethink the AI and make it a flying phantom that moves around the all map.

When we made the decision of changing the AI we hadn't implemented the jump mechanic for the player, at the time we had it we realized that it wouldn't be so difficult to add it to the AI, so we remade the AI once again in the first way we had thought.

Finally, as commented in the gameplay part, we received the feedback of the people and they told us that it would be funnier if we had more enemies to fight. We hadn't much time to make a lot of enemies so we decided to get the phantom back so we had two different enemies and finally we made variants to get four different enemies.

RENDER

Now, for the render part we had to try some different render methods to finally achieve a functional render with a high refresh rate to get the best and most dynamic gameplay experience.

First we implemented a method to delete the entities by drawing a box of background color and then drawing it in the new position every frame. Later we thought of adding a tilemap to have a pretty background in our game so we made it and we had to change the delete method of the entities. We tried to redraw the tilemap every frame but this was very expensive in terms of performance and it would affect the gameplay experience.

We continued wanting a pretty background so we implemented a method to redraw only the tiles affected by the entity in the previous frame, it was not satisfactory enough for what we wanted to achieve, so we had to give up the idea of having a pretty background in the game.

Finally we made the render method to be the first that we had implemented. Later, when we had all the mechanics and some different AI implemented we thought that we could fit a pretty background that only has things drawn in the superior part of the screen where the entities never arrive. Also we thought that if we used one color exclusively for the background we could change it when we wanted to give more personality to the different levels that we had.

Late Stages of Development

Once we had a playable prototype it was time to transform that prototype into a videogame. We stopped programming to focus all our effort on creating the art and de music that were necessary to reach a good final looking proyect.

ART

Characters

About the art, we created sprites for 2 players and 4 enemies. All of them include different animations for different states: idle, run, jump, die and shoot.

Although this is a fighting game, we wanted to keep an amusing art style, because of this we decided to make a mushroom (fungus) inspired character. Firstly, we created the Player 1 and then we modified it to create the other sprites.

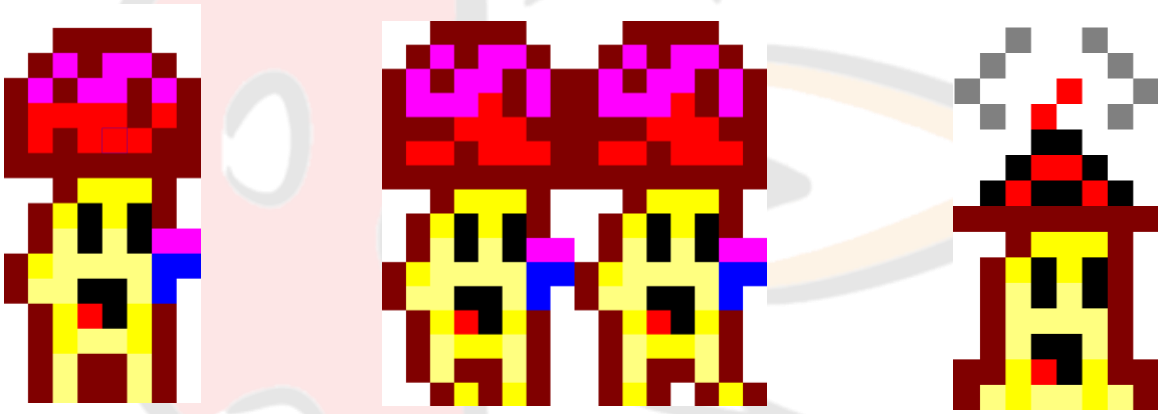


IMAGE 4: CHARACTER SPRITES

As a reference to the Amstrad CPC game 'Rainbow Islands' we added a rainbow to the die animation.

IMAGE 5: RAINBOW ISLAND REFERENCE

For the players' sprites we only changed the colour but with the enemies we went further. To express the different behaviour of each one and their level of difficulty we changed their colour, expressions and also their cap.

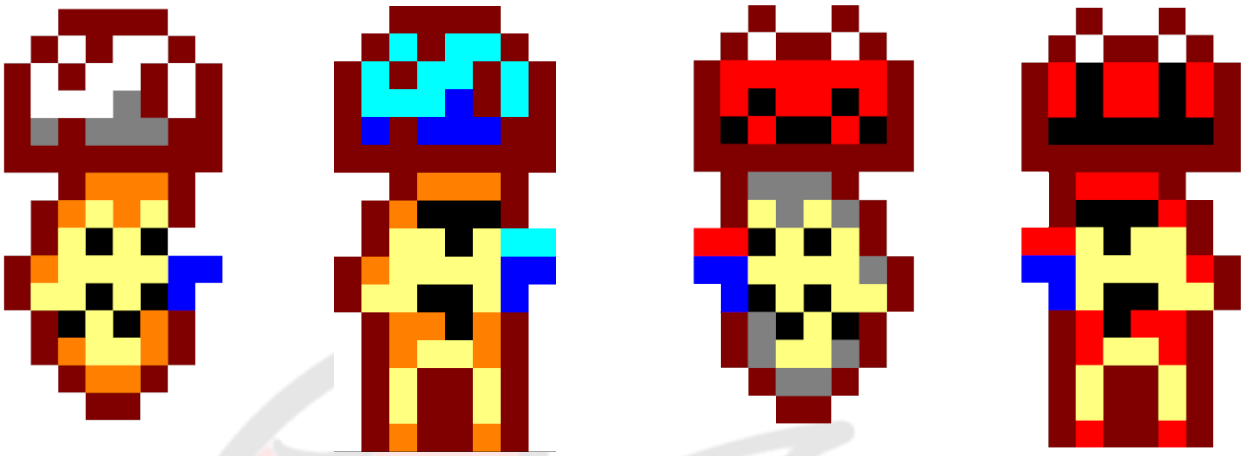


IMAGE 6: ENEMIES SPRITES

Bullets

As a way to notice the player that when he shoots he now controls the bullet, the bullet itself has waves. This reference pairs with the antenna and the waves of the shooting sprite of the player.

IMAGE 7: SHOOTING MODE SPRITES

Platforms

Secondly, we started adding the platforms to our game. As the theme is about fungus we kept that art style by creating grass and rocks platforms. As you get closer to the final enemy the grass decreases and the rocks get more importance in the scene. It is the same way with the floor. At first we only did static sprites but, taking proffit of our animation system, it was really easy to include animations to our platforms.

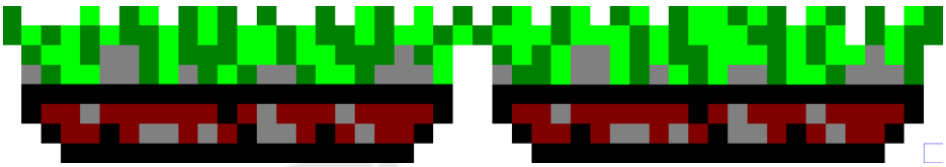


IMAGE 8: GRASS AND ROCKS PLATFORM

Environment

Finally, we added the backgrounds to the levels. It all was made with patience and inspiration. To express the progress to the final stage, the background expresses the pass of time as the day goes by. It helps the player to feel that the next enemy is darker and more dangerous than the previous one. Also, we know at which point our art abilities are, so we wanted to create a simple but nice looking art.

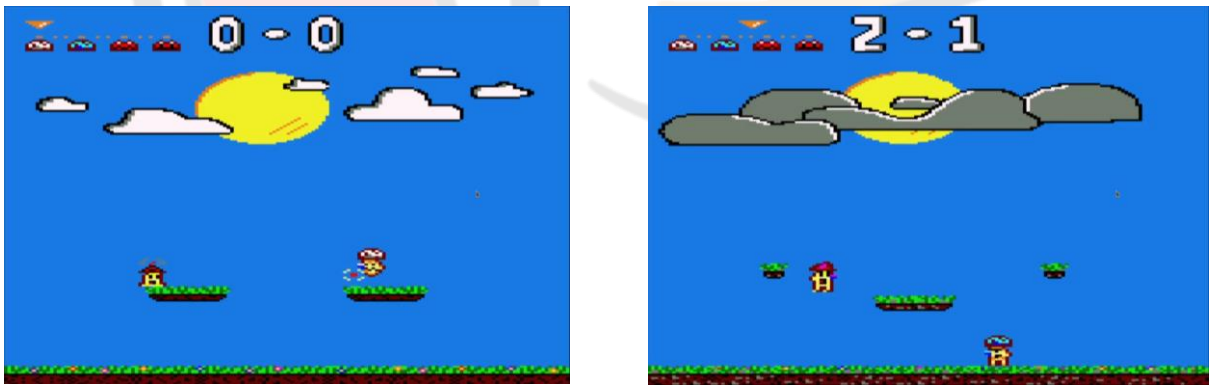


IMAGE 9: FIRST ENVIRONMENTS

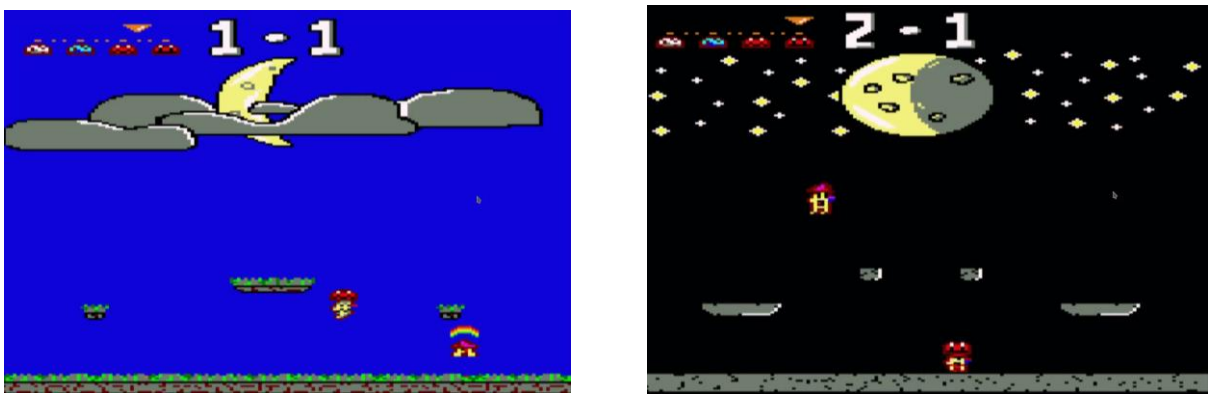


IMAGE 10: LAST ENVIRONMENTS

MUSIC

For the music, we wanted to create a dynamic and a bit of an aggressive soundtrack. Using the *Arkos Tracker 1*, a software that can be used to create music in .aks format, and a MIDI *Akai MPK mini* we reached exactly what we wanted. Because of using Arkos Tracker 1 and thanks to the CPCTelera's music utilities it was really easy to add music to our game.

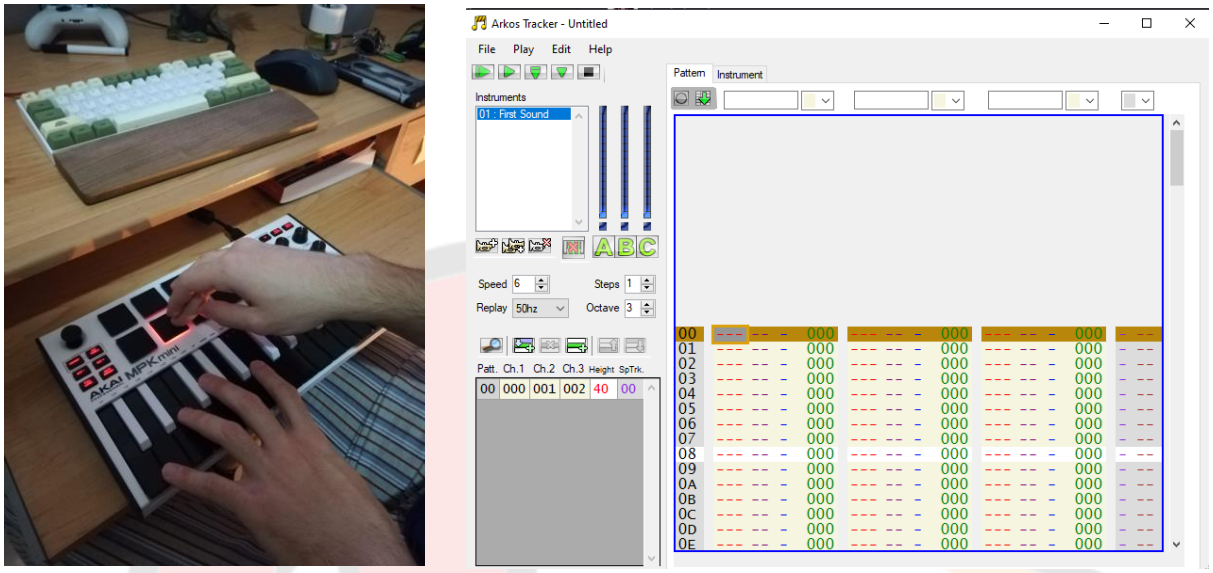


IMAGE 11: USING THE MIDI WITH ARKOS TRACKER

USER EXPERIENCE AND GAME FEEL

Also, another important point that we took care about was the user experience and game feel playing the game. As the game has a few controls we wanted to be sure that everyone understood them so they could enjoy the game to its maximum point.

Furthermore, at this point we wanted to make the game as smooth as we could, keeping a high frame rate and a good usage of the machine resources such as memory, registers, etc.

User Experience

As in any other project that is oriented to the user, we needed to test it with people that were not related to its development to know their opinion and point of view without any previous knowledge that could affect their perspective about the game.

Because of this we asked some classmates to test the game and to tell us their opinions about the controls, if the menu screens were understandable and if it was difficult to play. Those test

sessions were really useful for us to polish the game in every aspect: art, game feel and user experience.

Menu, messages and HUD

As our game has two game modes we needed to create a main menu screen with multiple options: the first one to select the single player mode by pressing the number 1, the second one for the multiplayer mode by pressing the number 2 and at last but not least, the controls screen.

IMAGE 12: MAIN MENU

The last option drives the player to another screen where he can choose between going to the player controls or the bullet controls. We decided to split the controls screen in two separate screens because adding a lot of information in one screen could confuse the player. Also, splitting it in two screens would help the player understand the mechanics of the game in a better way.

IMAGE 13: CONTROLS SCREENS

In order to tell the player that he is going forward to some objective we added a progress bar represented by a pointer, the enemies caps and a way of points. Each time the player wins a round the pointer advances one position, so this related to the score helps the player to understand that he needs to win 3 rounds to achieve the next enemy.

At last but not least, when the player wins a round, a message comes out to give the player a break to continue or to exit the game. Also, when the player loses against an enemy a Lose Screen is printed so the play knows what to do to play again. This is the same way when the player reaches the final enemy and defeats him, a Win Screen is printed.

Everything is made to tell the player what is happening in each moment.

Game Feel

About the game feel we did some improvements related to the collisions with the platforms,

the difficulty of the AI, jump force, level design and some optimizations of the code.

In the first place, we noticed that when a character collided with his head with a platform it was teleported to the top of it, so we changed it to only check the collision with the feet of the sprite. Also, when the player jumped into a platform if he did not stop pressing the key it gave the feeling of making a really huge jump. We modified it to need to stop pressing and to press the jump key.

In the second place, we improved our level design by testing each enemy at its own level. Because of this we could adjust the platforms so the level was more enjoyable for the player but also suitable for the enemy and his behaviour.

Once we had our levels designed we put them to test again with people that were not related to the project to see if it was easy, difficult or hard in order to avoid people leaving our game because it is too difficult. Thanks to this playtesting we managed to reach a good curve of difficulty and learning.

Finally, about the optimizations, we checked each system and its cycle time to see if they were as fast as we knew they could be. In order to optimize the render, when an entity is stopped we do not erase it and redraw it, we only redraw it.

Another huge optimization came by refactoring and reprogramming some macros that were made at the beginning of the development and that were used in almost all for_all loops in the updates of the systems.

Learned lessons

Making a game in assembler code gives you an incredible amount of lessons. Low level programming is always a good option to know how the gears of a specific machine or technology work.

These are some of the **most important lessons** that we've learned:

- **The best option usually is not the first one**: in order to achieve the best solution for a problem, you probably should think about it more than once. Working on *The Last Fungus was* an incredible experience to learn about this and refactoring inefficient solutions for better ones was our best acquired knowledge.
- **Debug data, not code:** what matters in your software is not the code, it's the data. You can say that your program is working if all the data is correct and it's working as you want. Debugging data instead of code was a helpful mindset. Besides, working on this kind of machines and languages, this is a highly recommended practise, talking from our experience.
- **Focus on the final product:** adding many cards to your backlog without having a final product can be overwhelming. The idea of focusing on the final product was always on our mind. Communication and tools for organizing the work give us the ability to develop a final product, focusing first on the tasks for the final product.
- **Communication, what a beautiful word:** any kind of team project without communication is intended to fail. It was a good tool to know what everyone was doing. Knowing this, unexpected results didn't happen, the ideas were crystal clear and everything was working fine. Letting your teammates know what you are doing and listen to them is the key to success.
- **Don't use things that you don't know how they work:** if you want your code to do what you want, don't use things that you don't know how they work. It sounds common and it makes sense but we usually do this. Knowing what your things do is extremely helpful in order to optimize your code and try to achieve better solutions.

We really appreciate the time you take to read this, thank you.

We hope you enjoy *The Last Fungus.*

Resources used for this documment

Press Start 2p – Text Font – By: Codeman38 – License: Released under the SIL Open Font License.