Baba's Palace

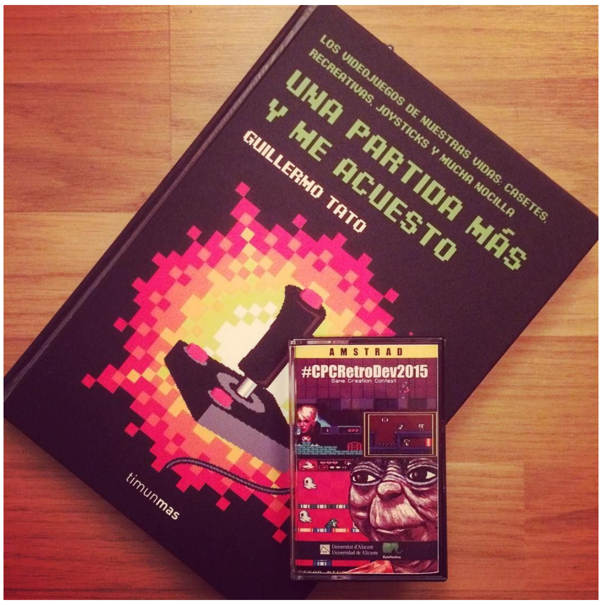
A 64Kb videogame for Amstrad CPC

Code & Gfx by Rafa Castillo · Music & Fx by John McKlain

MAKING OF ...

THE ORIGIN

All began in October 2016 when I went to Retroconsolas Alicante with attend the The Oliver Twins conference. In my childhood I had an Amstrad 464 and I have always captivated by video games for 8 bit systems. In Retroconsolas I realized that development for Amstrad was living a second youth, thanks to impressive productions homebrew like Bubble Bobble 4CPC, OutLaws, Magica, Hire Hare, Space Moves, Space Pest Control, Adiós a la casta, Frogalot, Golden Tail, La Guerra de Gamber, Imaginario colectivo, Teodoro no sabe volar, Phantomas Saga Infinity, Hair-Boy, TopTop…


LOS VIDEOJUEGOS DE NUESTRAS VIDAS, CASETES, RECREATIVAS, JOYSTICKS Y MUCHA NOCILLA
UNA PARTIDA MÁS Y ME ACUESTO
GUILLERMO TATO
timunmas
AMSTRAD
#CPCRetroDev2015

Was that day when I discovered the CPCRetroDev contest, and in that moment I saw it clearly: I will develop a game for my Amstrad, which many good times gave me when I was a kid. I'll participate in the next edition!

Once I was convinced of my purpose, I began to investigate how to develop a video game for Amstrad and find CPCtelera and Fran Gallego videos and tutorials. That was amazing! Can you ask for more?

Night after night I started to refresh my C knowledges (I've 15 years without programming in C) and study carefully all videos about programming CPCtelera published by Fran. Gradually I learn a lot of stuff with help of the examples included in CPCtelera pack. Soon I start to feel illusionated with the project and then started think about the game that I would like to create.

THE IDEA

Although I've had previous experience in game development, I was a complete novice about Amstrad, and I knew I didn't want get into a project with excessive complexity. So I get out the drawer an idea that was around my head for many years and that fit with what I wanted.

In 1985, Yutaka Isokawa coded in Hudson's Basic for Sharp MZ-700 a video game titled Pitman. In 1990 a version for Game Boy with the same name (a.k.a. Catrap) was published. It's a game that made me enjoy a lot in my youth, with mechanics from classic games like Boulder Dash or Sokoban. I found an ideal challenge develop a game based on this concept.

Looking back, I have to admit that I found it extremely hard this development. The concept of Isokawa obeys to a really complex logic, and a series of actions that happen in cascade (such as gravity application in particular cases) which required from me a bigger effort than I assumed at the beginning.

Learning

I does not take long time to realize the importance of a single byte when I only have 64kb. Shortly after starting the development I had to get to optimize, optimize and optimize: saving on gfx, reprogramming algorithms, rethinking how to include 100 levels, find and eliminate unnecessary waste of memory...

Step by step I was getting into a new way of thinking: how to get the most with the least. Each new idea should be subject to how much memory would be needed to carry it out. The accounts of the memory that I need should be very clear: How much the music takes, how much the definition of the levels is carried, how much the graphics, the logic, the different options...

And here came to me the most surprising thing: it was very funny! I enjoyed thinking how to scratching a few bytes here and there, giving a turn to an algorithm, or planning how to save a certain step, or how to improve a concept to optimize performance or in memory. It was like having a giant puzzle to plan thoroughly how to solve it.

THE PROPOSAL

With the first image of the game I made public, I got the attention of McKlain, one of the best musicians in the Amstrad scene, with a long list of video games in his portfolio. A really number one! From that moment I saw it clear: I should to excite McKlain so that he participated in this development. I knew that this would be as my first project for Amstrad, and the proposed collaboration must be more than words, so I decided to create an attractive prototype before to launch a serious enough proposal to McKlain.

In March I got a development level that allow glimpse what could be the game. I create a teaser video presentation of the project and privately I launched the proposal to McKlain. In five minutes I had his answer in form of emoticon, a hand with thumb up. Wow!

In late March, when I presented Baba's Palace with the video teaser in social media, I was able announce that the very same McKlain would handle the music section. For me, to have McKlain in this project, his talent and his experience, is a dream come true.

Their collaboration also meant for me a great responsibility. Now there was another person involved in the project and I could not fail.

The challenge

For me, the development of Baba's Palace has meant a real challenge due to my technical ignorance of Amstrad and my so rusty knowledge in C. But regardless of these circumstances, and specifically regards with this game, I found with a major challenge in the following aspects:

Graphics

I oopted for a play area that occupy the entire screen, which in practice means abandoning the use double buffering and manage all sprites drawing controlling raster location to avoid flickering. The complexity of this grows when the goal is to animate up to 24 characters on screen without any loss of performance.

Levels

Create a game with 100 levels has been a huge challenge due to the necessary memory optimization, from the great effort to come up many levels, to the creation of a level editor that allow their testing in real time. Without taking care all these factors, get 100 levels for this game would be almost impossible.

Logic

One of my biggest surprises for me was how a game, with a relatively simple aspect, could involve a high logic complexity. The different cases that can happen playing this game are really complex, especially if we consider chain events when player push stones or walk to a different position with stacked items over himself. There is a case where a single player movement involves a sequence of 5 events, 2 of them with cascade falling items. Crazy!

Rage against the memory machine

The need of optimize has been the general trend during development. I dare say that for every hour destined to code, I spend 2 hours re-coding algorithms and functions, all to save the maximum in bytes.

THE VIDEOGAME

There are two main ideas related to the artistic conception of this game:

1. **BABA YAGA**
   Baba Yaga is the name of a famous witch from Slavic folklore and mythology. It is over a magic flying mortar. I’ve been always fascinated with this character...

2. **KUNG FU**
   TV series of mid-70s starring David Carradine who narrates the adventures of a Shaolin monk traveling solo through far west as unique weapons using their skills in martial arts and inner strength of his philosophy of life: Buddhism.

As a result of these concepts arises the game story, a little anachronistic, where Little Indian and Shaolin Kid's face Baba Yaga henchmen in their search of the elixir of life, to get the father of the little Indian come back to life.

Baba's Palace is a logic game with arcade aesthetics and features up 100 levels. The player can handle Shaolin Kid and Little Indian to solve puzzles eliminating all enemies presents at each level, and descending to the depths of the Baba’s Palace in search of the elixir of life.

Gaming experience

For me has always been very important that the player feel very comfortable playing. Therefore the option of playing with joystick, keyboard and even to redefine keys is provided on the main menu.

As for the movements of the main character, it has been sought by all means that the action has a lot of rhythm. It would be desperate to mentally visualize the route that you want to perform with your character and that the journey is insufferable by slowness in the actions.

The first levels serves to introduce the player into the game mechanics. Like a tutorial, the game will expose the main rules governing logic, so the player learns, almost without realizing, that the stones can be pushed, the enemies are eliminated by touching them, there are walls that can be destroyed, etc.

The creation of the 100 levels has been quite complex, especially because to devise a level representing an intellectual challenge have a lot of work and has to pass a rigorous quality control. Five of the final levels are adaptations of levels that can be found on the Game Boy version. They are mythical levels and deserve be known and enjoyed.

The most common failures in creating levels are:

1. Create a level that is impossible to solve. To avoid this you must try every levels.
2. Create a level that can be solved in a simpler way than the idea under which it was initially developed. In order not to fall into this error, many tests must be done to ensure that the only way to solve the level is to follow the strategy with which it was initially designed.

When the player overcomes a level, a small fanfare (success jingle) sounds, reinforcing the message of LEVEL CLEAR and rewarding the player for his good work ..

Technology

In order to achieve the desired rhythm of play it was very important to get a very efficient sprites draw routine. When not using double buffer, it is necessary to manage sprites very well to avoid flickering. Here a technique is applied that has allowed me a very important improvement of the performance.

In front of the classic waitVSYNC (which also does not get rid of the flickers in the upper screen area) I have implemented a function called waitRaster. This function allows me to know the screen area where the raster goes. If the raster is not in the draw area, I draw the sprite directly. If I detect a collision, a brief wait is made to ensure that the raster has passed through the problem area before draw my sprite. The performance improvement against waitVSYNC is spectacular!

Another very delicate point is the amount of enemies that can be simultaneously on screen. The implemented system allows up to 24 enemies on screen, changing status every 0.5 seconds (idle movement). Again, thanks to being able to paint sprites without having to do a waitVSYNC this has been possible at the desired speed.

The transition between screens is done by fades to/from black. It is a very quick effect, but gives a very interesting elegant touch. The palette goes through 6 different states to make every fade, and the colors have been precalculated by hand to achieve the desired effect.

Draw text on screen function lets draw text in different colors, and handles uppercase, lowercase and variable spacing between characters as in the case of the letters like i, l...

In the intro has been used *Exomizer 2 decoder Z80* in order not exceed a limit for acceptable Kb intro (here 8kb).

One of the biggest technical challenges I have faced has been how to encode 100 levels so that they are visually attractive and with the lowest cost in Kb per level. In order to overcome this challenge, I have developed a system of coded levels with a low number of elements to later procedurally create an attractive final level.

- Each level consists of 11x9 tiles, but the last row is always a solid block, so I only need to encode 11x8 tiles.
- Each level has up 8 different types of tiles: solid block, destructible block, ladders, ground enemy, flying enemy, heroes characters, stone and void. So use 3 bits to encode each of these elements.
  In total, 11x8x3 bits = 264 bits = 33 bytes per level → 100 levels = 3,300 bytes.
- To load a level, I read the type of element in a certain position and, depending on the elements around it, select the graph that should be assigned to that position.
- In the case of painting a background tile, depending on whether we are in a even or odd row, one type of graphic is selected and, depending on chance, if that graphic is its light or dark version. Everything destined to break feeling of pattern or repetition.
- Finally a Beautify function is executed that improves the graphical aspect of the level with columns and faces of old idols.

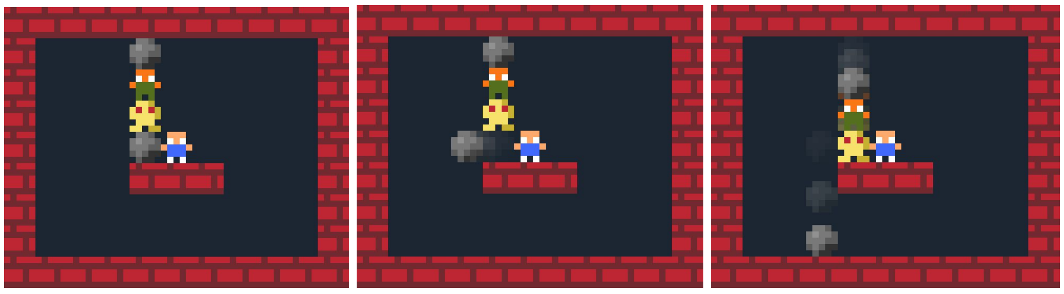
Shortly after working on creating levels I realized that the system I was following to create the level with a classic level editor, compile the project and test it in the emulator to then detect errors or improvements and start over was not viable. Finally, I ended up developing an own level editing system in PuzzleScript, which allowed me inside the same level editor to play level and detect problems and improvements in real time. This has been key to being able to undertake the production of my highly desired 100 levels.

Logic and Artificial Intelligence

This project has no enemies that present an intelligent behavior based on artificial intelligence algorithms. On the other hand, it has a very advanced logic programming to solve the great number of situations that can occur when interacting with the elements of a level, as well as to procedurally generate each level.

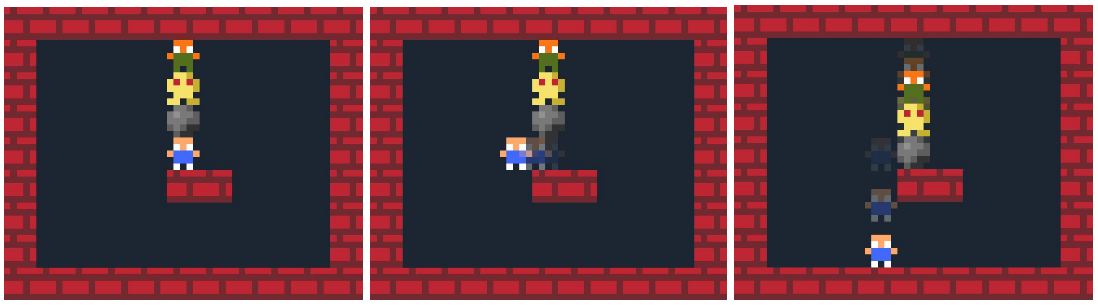
The player can interact with level elements in the following ways:

- **Pushing stones**
  When pushing a stone, it moves to occupy the adjacent hole according to the direction in which it is pushed. All elements that may be affected by gravity begin their fall in cascade:
  - The same stone we just pushed if there is no floor in the adjacent position.
  - The elements that were on top of the stone (other stones, terrestrial enemies, companion protagonist).



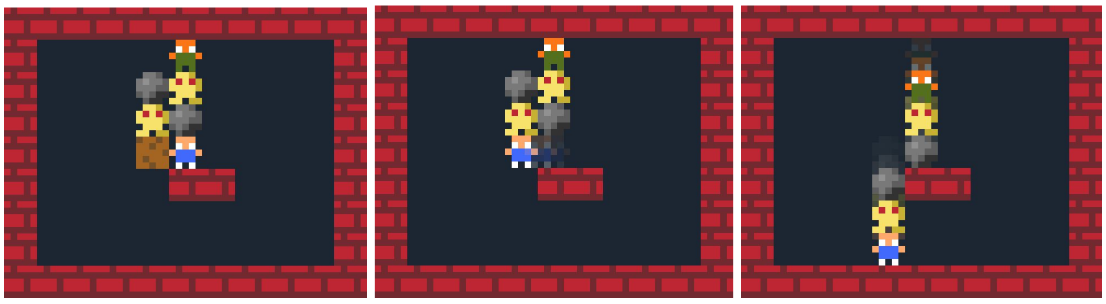
- **Walking to an adjacent position or down the stairs.**
    - It can give the situation where the protagonist walks to an adjacent position, but on the protagonist may exist elements that are affected by gravity.
    - In the same way it can happen that the protagonist goes down a ladder having elements affected by gravity on himself.

  When performing any of these actions, the elements affected by gravity would cascade to the position occupied by the protagonist.

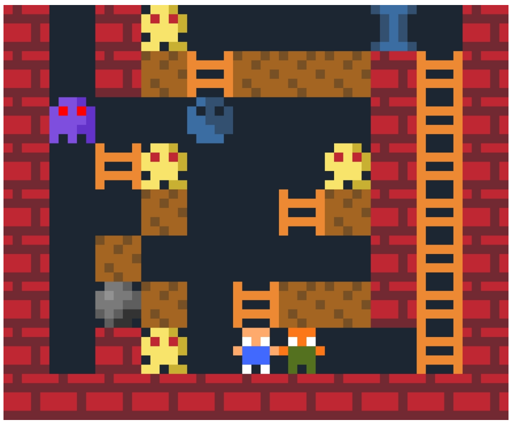


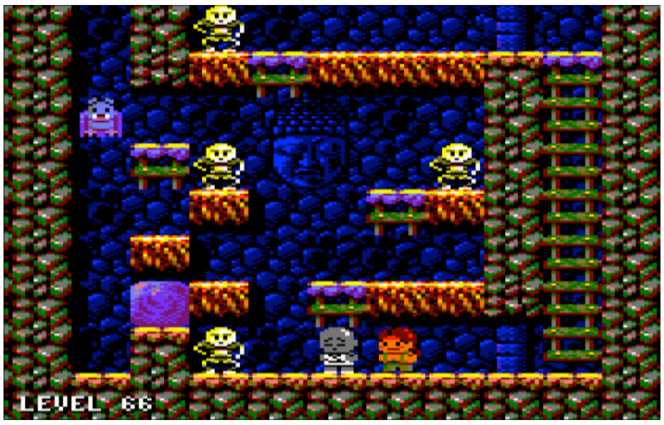
  The most complicated case occurs when the protagonist walks to an adjacent position, breaking a destructible wall or eliminating a flying enemy, while in that adjacent position there is no floor, which would cause:

    A. The fall of the protagonist.
    B. The fall of all objects affected by gravity that were on the destructible wall / flying enemy.
    C. The falling of all the objects affected by gravity that were on the protagonist before beginning its passage to the adjacent position.
    D.



For procedural generation of levels, logical algorithms are combined to select the appropriate tile depending on the type of tiles that surround it, and random functions conditioned to a certain logic to achieve variety in the backgrounds like columns and faces of ancient idols.


PuzzleScript level editor



Real level inside the game

Graphics and design

Graphics and character design are inspired in late 80s and early 90s anime series.
Each of the protagonists has up 20 different frames to undertake all actions performed in the game such as walking, pushing stones, destroy enemies, climb ladders, jump from platform to stairs and back again, jumping from one ladder to another, pushing stones from ladders...

Special care has been taken in the treatment of accelerations and decelerations in the animations. This can be seen, for example, when pushing a stone, hitting enemies or going down stairs. They are actions without uniform movement, but with deceleration, a detail that helps to make attractive the actions of the protagonist.

The anticipation of the movement has also been taken care of. For example, before hitting an enemy, pushing a stone or climbing stairs, the protagonist assigns a few frames to prepare his action, which reverts to more "realistic" and less "robotic" animations.

I opted for a typographic font with case sensitivity, choice colors options and with different letter widths (as the case of the 'i' or 'I') to present more attractive texts.

Finally, a special treatment of backgrounds is done to reduce repetitive patterns and give more variety to areas that may occupy a large size. To accomplish this we have differents tiles for even and odd rows, and 2 for each tile versions: the illuminated and dark version. In addition, columns and idol faces are incorporated with the same purpose: get more variety and less sense of repetition.

Music and FX

In this game you can listen up to 4 songs:

- Intro
- Main menu theme
- In-game theme
- Clear Level fanfare.

All the melodies have been worked with special attention to achieve coherence between all of them, and that together convey a unique and particular style.

In-game theme music has been treated in a particular way to allow leave a free enough channel to not interfere with the sound effects.

The game has 10 different sound effects that are executed when pushing stones, blocks sand are destroyed, falling item landing, jumping actions, etc. towards getting a full game experience.

Of course, saving memory has also influenced this development, which in successive iterations has worked the instrument definitions and structure patterns to achieve maximum bytes savings with minimal damage in quality.

Tools used

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

- CPCtelera 1.4 - Astonishingly fast Amstrad CPC game engine.
- WinAPE 2.0b2 - Amstrad Emulator.
- Sublime Text - Code Editor.
- Aseprite - Sprites and loading screen creation.
- Arkos Tracker - Music and FX creation.
- PuzzleScript - Level creation and quality control.
- Trello - Tasks manager.
- Google Drive - Storage of sources and materials.
- BitBucket - Remote Code Git repository.
- SourceTree - BitBucket code Git repository manager.
- BabasLevelParser - Own tool to encode levels with the code generated with puzzlescript.
- MemoryMapCalculator - Own tool to calculate the location of each asset or data graphic in the lower memory.
- Google Docs - Generation of documentation.

Credits

Code & Graphics: Rafa Castillo (@azicuetano)
Music & Sound Effects: John McKlain (@elmcklain)