MAZE ADVENTURE

Albert Sirvent Jerez

Gameplay

Game Controls

- Directional arrows: Move through menus.
- Return: Select menu entry.

Main Menu

New game
Start a new adventure.

Load game
Load a previous game, you need to insert a valid code.

Options
Here you can turn on/off the music and the textures (turning off the textures will make loading times shorter).

Credits
Who made this game.

Ingame UI

Level
Indicates the current level of the game, you must reach level 32 and defeat your last opponent to complete the game.

Stats
Here you can see your current HP, your attack and defense values, your current potion and scroll amount and if you have the key needed to advance to the next level.

Log
The log displays what happened in the lasts turns (damage taken, damage done and items collected).

Compass
Indicates where is the North in the 3D view.

3D view
What your character see.

Minimap
A top view of the maze, it only displays your surrounding area and enemies within this area.

Ingame Menu

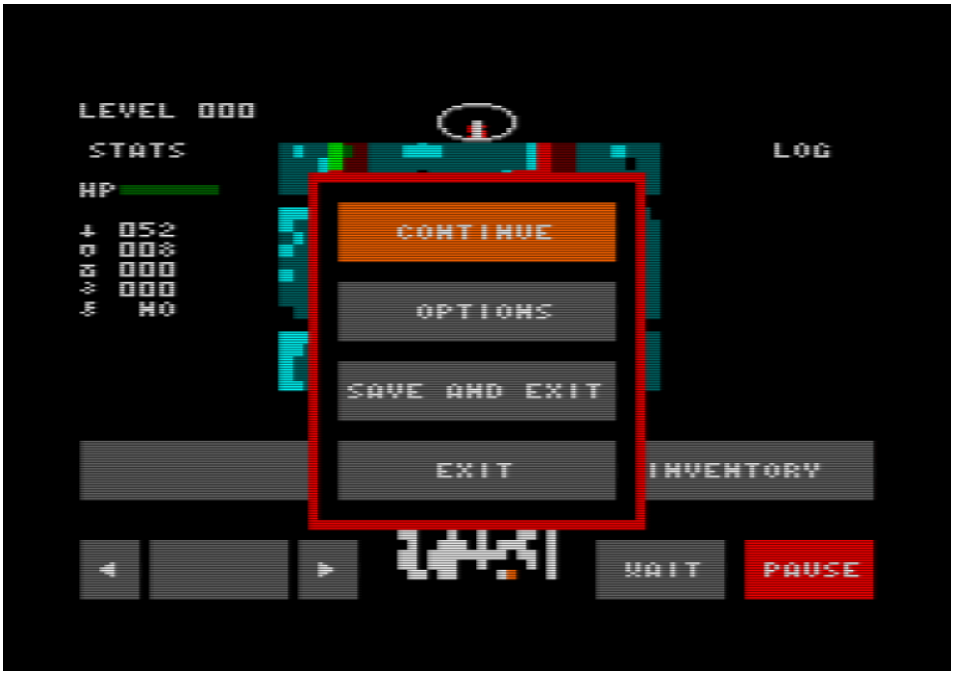
Action button
Displays the current possible action, it can be attack an enemy, pick up an object or go to the next level. When selected your turn ends.

Turn buttons
They make you turn right or left. This doesn't end your turn.

Inventory
Open the inventory menu, drinking a potion or using a scroll ends your turn.

Wait
Ends your turn.

Pause
Opens the pause menu.



Pause Menu

Continue
Return to game.

Options
Open the options menu.

Save and exit
Show you your save code. Return to main menu if you press any key.

Exit
Return to main menu.

Motivation

For this edition of CPCRetroDev contest I wanted to develop something challenging, so I decided to make a 3D renderer for the Amstrad CPC 464.

Development

Renderer

So I started to develop the renderer. At first it was based on Wolfenstein3D renderer, using raycasting. Soon I realized that it wasn't a good approach, because it was really slow.

After that I started with the new algorithm for rendering, this is the one I used in the final game. It calculates what cells are in view and stores them in an array (maximum distance for vision is 6). Then it reads this data and draws first farthest walls and items to a buffer. When it finish draws this buffer in video memory.

All is quite hardcoded, so it only works in mode 0 and 80 by 100 pixels.

The game

When I had the renderer working, I started to create the game around its capabilities. I thought a dungeon crawler or a rogue-like would be perfect for this renderer. Those game genres are turn-based and don't need real time rendering so, at the end, I decided to make a simple dungeon crawler.

Maps

When I started to develop the game, I was developing and testing some map generators in Unity, so I decided to implement one in this game. A map generator gives me infinite maps and additional replay value, and it makes the final binary smaller as it generates all at runtime.

The first problem I encountered porting the generator from Unity to Amstrad was the dynamic memory allocation, it was making use of lists. So I tried to replicate this behaviour using big arrays. During the map generation I have a big chunk of RAM available, so I could implement the generator without major problems.

Enemies

At first, the game was designed with 4 different enemies per level. But as the development progressed the memory available decreased and I had to reduce to 2 enemies per level, the rat (common in all levels) and a theme based enemy.

With the AI happened something similar, at first there where 4 different behaviours for the enemies: Passive, aggressive, tactical and shy. Now only remains the first and the second linked to the rat and the theme enemy respectively.

The passive enemies don't attack you, the only roam the map, but if you attack them, they will attack you until their life is low, when their HP reach less than ¼ they will flee.

The aggressive enemies will chase you at first sight, and will attack until death.

Tactical and shy enemies were removed. Tactical enemies would only attack you when you had low HP or were surrounded by other enemies. Shy enemies would flee when they see you.

States

The game is organized in different states. To manage this states, I've implemented a struct that contains pointers to their functions. So the state manager only needs to know which is the current state and call its functions from the array. This way I can implement new states relatively quickly.

Game Credits

- Coding and idea
  - Albert Sirvent Jerez
- Textures and art
  - Alejandro Padilla Lozoya
- Music
  - Carlos Blaya Cases

Developed using CPCtelera.

Music created with Arkos-Tracker.

Textures made with Paint.NET and GIMP.