
PyCAPGE
Python, Classic Adventure,
Point and Click Game Engine

garba

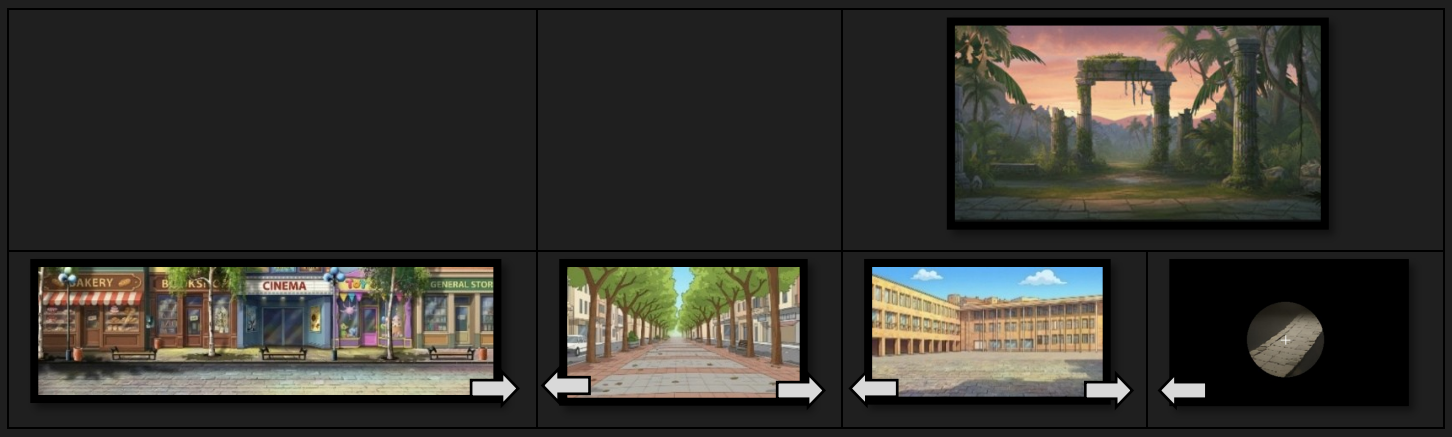
1. A QUICK LOOK

This version uses standard industry terminology (like "walkthrough," "mechanics," and "template") to sound like a professional software manual.

Overview This mini-adventure provides a quick walkthrough of the Engine, showcasing the full range of configurable features and combinations across just a few screens. The demo consists of five distinct scenes demonstrating different environments: static, scrolling, parallax effects, dark zones, and more. Each scene incorporates classic graphic adventure mechanics, including: Inventory management (picking up, using, and combining items). NPC interaction and dialogue systems. Map usage and character switching. Logic examples (conditionals and variables). Interactable machines, animated objects, and cutscenes. By providing a working template of these actions, programmers can easily adapt the code to create their own custom gameplay. This manual will guide you step-by-step through implementing each feature.

Requirements:

- **Programming Level:** BASIC
- **Graphic Design Level:** INTERMEDIATE

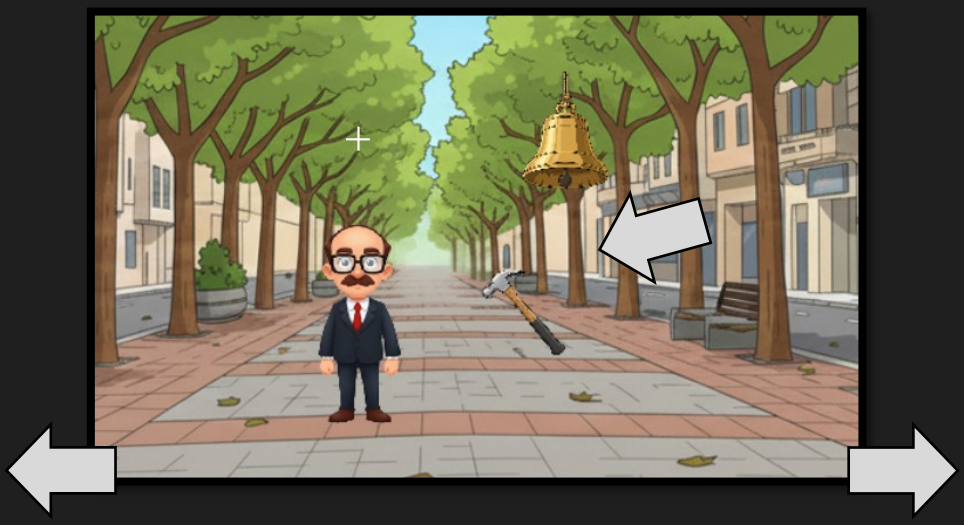


Demo Map Layout

Four screens are accessible by walking; one is accessible only via the map.

1.1 SCENE 1 – PEACE AVENUE

This is the home screen, a simple screen where an object appears—Hotspot—which we pick up and use with the bell. You will hear the sound of the bell and the sound of the first puzzle completed..



All backgrounds in the game consist of at least two images. The image you see on screen and a black-and-white image that indicates safe walkable areas..

1.2 SCENE 2 – TOWN HALL

There are several important elements in this scene.

1.- At the beginning of the scene, the NPC automatically travels to the center. A cutscene plays (it can be skipped with ESC and only plays once). It indicates that he was here 25 years ago: this is a nod to another graphic adventure I made in 2001 that began in this location.

2-. At the top left there is a Hotspot that is not loaded but generated by coordinates..

3.- Below is the large press that only works when the character says "use," creating its animation.

4.- On the right, there is a small press that is continuously animated.

5.- The next Hotspot is a lantern that we will take to see the dark room.

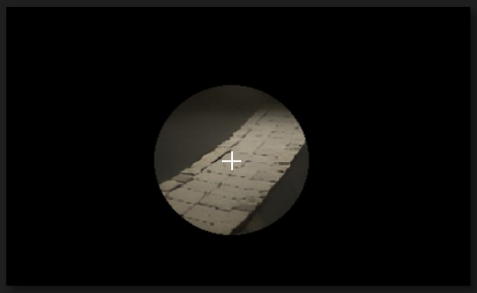
6.- The following object is an animated object that we cannot pick up, but if we pass close to it, the NPC changes color. It can be used for bonfires, dark corridors, etc.

Composition. Main, Walkable, Scene of NPC changing palette by Bonfire.

1.3 SCENE 4 – DARK ROOM

If we move to the right and enter without the lantern, we will find ourselves in a completely dark room. We take the lamp and enter again.

1.- With the lamp, we can see the room. We will have an adjustable light source.



2.- We take the map and use it: The map is designed to be truly useful. Apart from taking us to remote places, we can immediately go to certain scenes.

1.4 MAP USE

1.- We use the map to go to the Parallax Scene, which is the only one that is not accessible on foot.

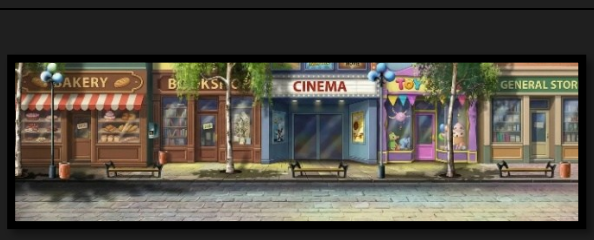
1.5 SCENE 3 – PANORAMIC STREET

The first peculiarity of this scene is that it has horizontal scrolling. You can walk down the street without abruptly jumping to another image. And since it's a long scene, we'll take advantage of this to put several elements in it.

1.- The NPC known as "Garba." We can talk to him, and depending on whether we have any items he needs or if we have spoken to him before, he will tell us one thing or another. In this case, he asks us for a flashlight with charged batteries.

Composition of the scene:

The system automatically detects scrolling, and the speed can be modified..



2.- Composite object – Crafting: "I will take the flashlight without batteries + the batteries" and, using them in the inventory, I will create a flashlight with batteries that I will give to the NPC "Garba." The conversation will be different if I have the flashlight with batteries than if I do not have it.

3.- The next object we have is a Hotspot with a different animation and assigned state. Here it is a trash can, but it can be used for anything, doors, levers, etc. It has the states open and closed. It is a Sprite Sheet with two images.

4.- Animated object with its own path. In this case, a beautiful black crow flies across the screen against the scroll. With this format, you can add clouds, rivers, characters moving frantically from one side to the other, vehicles, etc. These are Sprite Sheets with as many images as you want, where you can define the time each sprite stays on screen as well as different paths.

5.- There is another object that our main character cannot pick up because it is quite heavy, so he will have to switch with another NPC that we control in order to pick up the shovel and swap it. This is how we use this scene to explain the character swap.

We approach the other character and click on the verb USE. At that moment, we can control "Bart," who in this case is just our character wearing a beautiful red flannel suit. With Bart, we take the object SHOVEL and add it to our inventory.

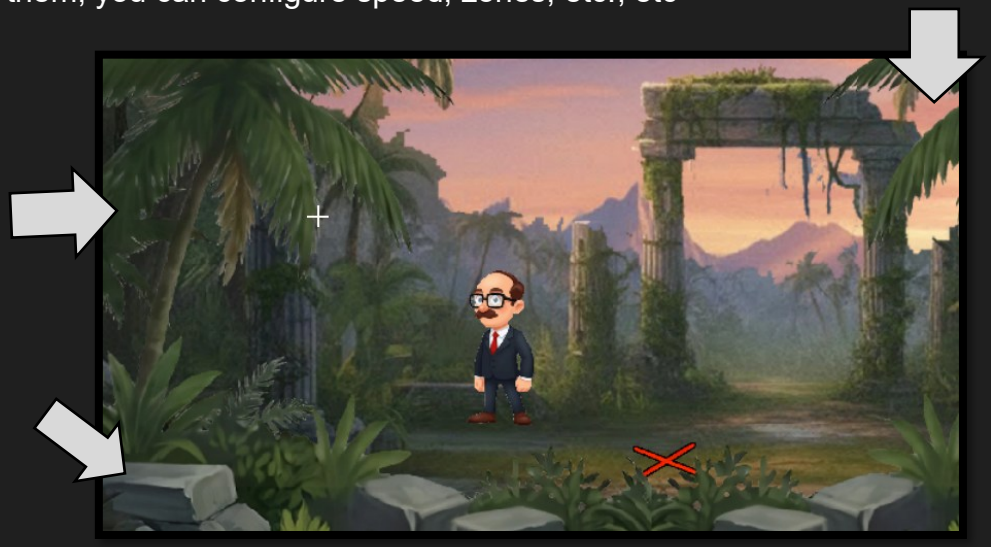
You can create as many NPCs as you want, and you can give them the freedom to move around all the scenarios or restrict them.

1.6 SCENE 5 - PARALLAX EXAMPLE

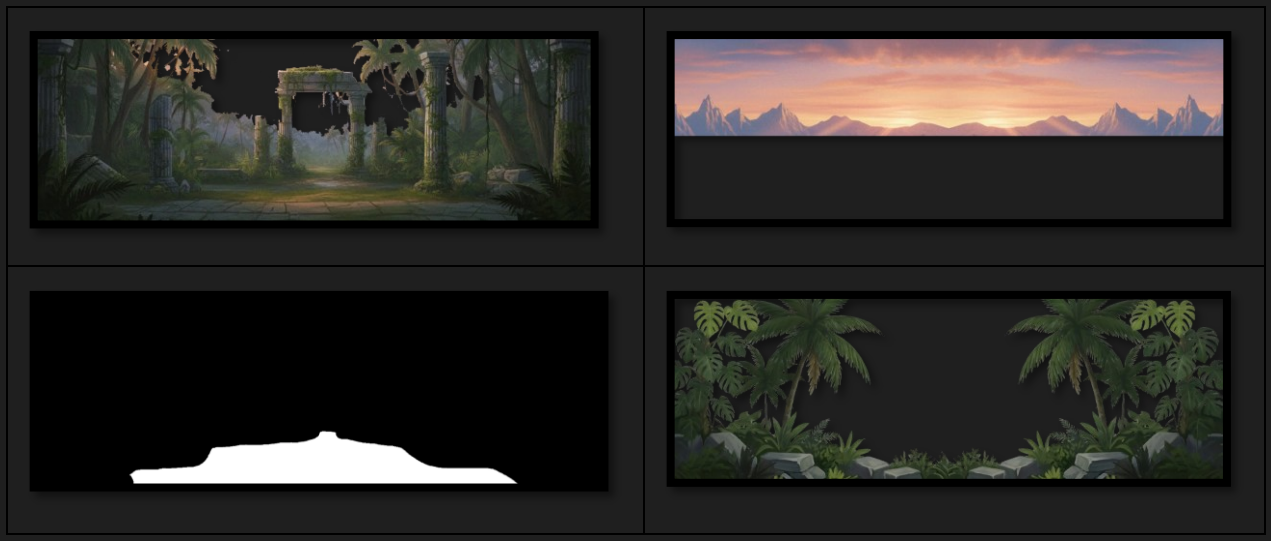
This scene is configured to use the Parallax effect and give more depth. In case you need to use it in different scenes. It can also work with automatic scrolling of the sky or background layer. It consists of 4 images. Which are:

- rear or sky view (automatic or manual when walking). FAR
- middle image or main image. MIDDLE
- Image indicating the walkable area. WALKABLE
- Closer image. Automatic with speed adjustment. NEAR

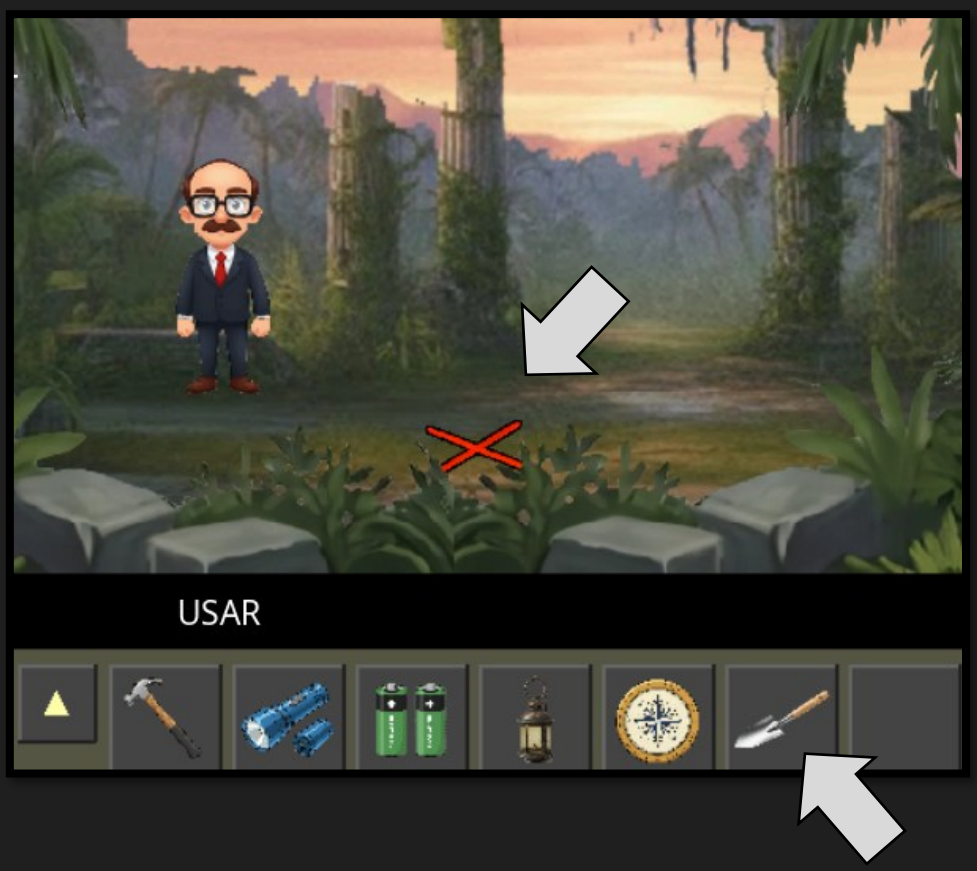
In all of them, you can configure speed, zones, etc., etc



The composition is made up of four images.



This scene also marks the end of the demo. Using the shovel on the pirate "X" takes us to the "ending" animation.



2. TITLE SCREEN

Home: it will first quickly load the initial images and sounds, which will be released from the RAM when they are no longer in use.

- NEW GAME
- LOAD GAME
- LANGUAGE
- CREDITS
- EXIT

The first step is to choose your language, and if yours is not listed, it's as easy as duplicating any file in the /languages folder and translating it.

We started with 57 languages available in this version. We took great care with this issue because it was important to us that the Engine be available to everyone. These are the 50 most widely spoken languages on this diverse and cool planet.

To do this, the system uses an "open source" font with all possible characters, and does not need to load any more, making it independent and functional on any platform and in any country.

The system is also programmed to be translated "on the fly," allowing games created in other languages to be loaded, as well as adapting all texts, verbs, and functions to different fonts, calculating sizes and reducing them until they fit in the right place..

This took me a week of work..

PS. Yes, before you ask, it's obviously translated into Klingon.

3. GETTING STARTED

Before writing a single line of code, we highly recommend **tinkering with the default script**. Combine this with your draft designs and text to build a mental map of what you need and what the engine can do.

Heads Up on Resolution Be careful when locking in your initial settings. Changing core configurations—like screen size—later on requires a lot of backtracking and extra work. While the engine is optimized to handle scaling up to fullscreen, defining your target resolution early is crucial. Remember: you will need to position objects, NPCs, and animations based specifically on your game's resolution.

Asset Recommendations We recommend starting with a **high-resolution canvas**, even if you plan to display it differently later.

- **Fonts:** The engine renders text twice (standard and HD repainting) so it stays crisp at any resolution.
- **Images:** We use advanced mathematical models to scale images, but they aren't magic. It is always better to start with high-res assets and scale down than to try and scale up.

Graphics & Text Once your story is written, the graphic designers take over. This is where the real heavy lifting happens. Since AI can't manage sprite sheets "yet," you'll need to do some old-school image editing.

Finally, **write all text in your native language first**. Don't translate until the game is finished and polished. We use **YAML files** for localization because they are much cleaner and easier to work with than JSON.

4. FILE STRUCTURE

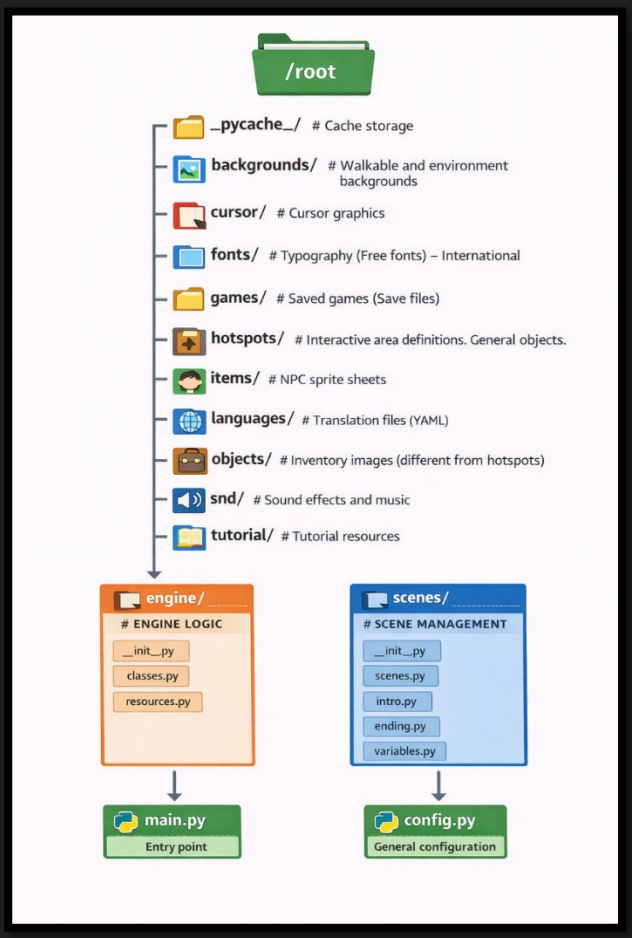
The system features a robust, **error-proof architecture** that keeps scripts and graphical assets in separate directories. This ensures that every component remains **independent and decoupled**.

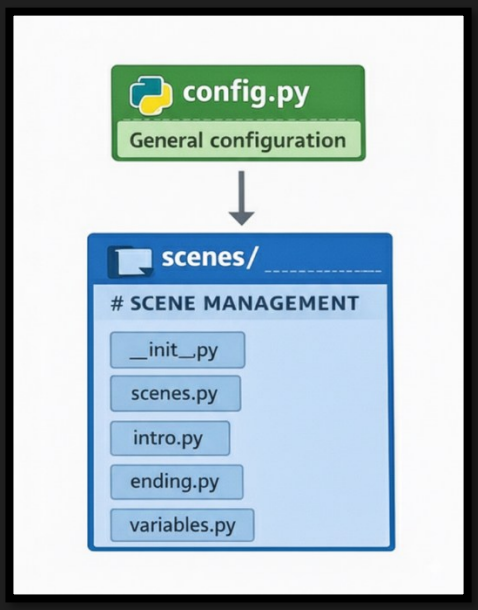
The **Engine's core** should remain untouched, as all configurations and scenes are handled externally. That said, since this is a fully **open-source** project, advanced developers are free to tweak the source code to add custom features or suit their specific needs.

If you develop an improvement that could benefit the community, feel free to submit a **pull request**! PyCAPGE is hosted on **GitHub** for distribution, updates, and community contributions.

Directory Overview:

- **__pycache__** → Python bytecode cache.
- **\backgrounds** → Scene backgrounds, walk-maps (walkable areas), etc.
- **\cursor** → Custom cursor assets and styles.
- **\engine** → Core logic—don't touch this unless you know what you're doing! XD
- **\fonts** → Font files (including global character support).
- **\games** → Saved games in JSON format.
- **\hotspots** → Interactive on-screen elements.
- **\items** →NPCs and main character assets.
- **\languages** → Automatic localization files (YAML).
- **\objects** → Inventory items (distinct from Hotspots).
- **\scenes** → Your game scenes (intro, ending) and global variables.
- **\snd** →Audio files and sound effects.
- **\tutorial** → This manual you are currently reading.
- **config.py** → The main configuration file. Feel free to edit!
- **main.py** → The primary bootstrapper. Don't edit unless you're absolutely sure.
- **readme.txt** → The usual stuff.





In theory, to create your graphic adventure at the code level, you will only need to work with these files: the general configuration, the intro, the ending, the variables you use at the global level for completed actions/puzzles, and the scenes in your game..

5. ENVIRONMENT SETUP

The Setup (The part everyone forgets):

- Install Python 3.x: Make sure you have the latest version of Python 3 installed.
- Install Dependencies: Run the following command in your terminal: pip install pygame pyyaml
- Running the Game: Simply double-click main.py or run it from the console using:

python main.py

6. DEVELOPER CHECKLIST

STEPS TO CREATE A NEW SCENE:

- **Assets:** Create the background image (bg.png) and its corresponding walkmask (bg_bm.png) in the /Backgrounds folder.
- **Config:** Add the scene name to the configuration file.
- **Coding:**
- Instance the Scene class within scenes.py.
- Define on_enter (for background music) and on_exit logic.
- Add **Exits** (pathways to other scenes).
- Add **Hotspots** (interactive objects).
- **Registration:** Add scene_manager.add_scene(my_new_scene) at the end of the file.
- **Testing:** Use **Debug Mode** to fine-tune the hotspot coordinates.

7. GLOBAL VARIABLES

```
# ==============================================================================
#  DEVELOPER MANUAL: GLOBAL STATE MANAGEMENT (variables.py)
# ==============================================================================

# The GAME_STATE dictionary is the object that the engine saves and loads
# when the player saves the game. Everything inside here is persistent.

GAME_STATE = {
    # ------------------------------------------------------------------------
    # 1. OBJECT STATE (INVENTORY AND WORLD)
    # ------------------------------------------------------------------------
    # These variables control if an object has been picked up from the ground.
    # Rule: If 'False', the object appears in the scene.
    # If 'True', the object disappears from the world (because it is already in the
inventory).

    "campana_recogida": False,
    "martillo_recogido": False,
    "farol_recogido": False,

    # Example of combined states:
    "pilas_linterna_recogidas" : False,    # The batteries separately
    "linterna_sin_pilas_recogida" : False, # The empty flashlight
    "linterna_con_pilas_recogida": False,  # The result of combining them
(Crafting)
    # ------------------------------------------------------------------------
    # 2. STORY PROGRESS (FLAGS)
    # ------------------------------------------------------------------------
    # Used to know if the player has already seen an event or activated something.
    "intro_ayuntamiento_vista": False,
    "pala_recogida": False,

    # ------------------------------------------------------------------------
    # 3. CHARACTER CONTROL (MULTIPLAYER / NPCs)
    # ------------------------------------------------------------------------
    # These variables define who the "playable" character is at this moment.
    # The engine uses this to swap control between characters.

    "controlando_gilo": True,  # If True, the player moves Gilo.
    "controlando_bart": False, # If True, the player moves Bart.

    # Note for the programmer:
    # When changing these variables at runtime, the engine will hide
    # the "NPC" version of the character and activate the "PLAYER" version.
}
```

8. LOCALIZATION FILES

We have opted for the YAML format over JSON..

```yaml
#=============================================================================
#  TRANSLATION: ENGLISH (en.yaml)
#=============================================================================

language_name: "English"

#=============================================================================
#  VERB SYSTEM - VERB BOX
=============================================================================
verbs:
  WALK: "GO"              # <--- KEY IN UPPERCASE
  OPEN: "OPEN"
  CLOSE: "CLOSE"
  PUSH: "PUSH"
  PULL: "PULL"
  PICK UP: "PICK UP"
  USE: "USE"
  TALK TO: "TALK TO"      # <--- KEY IN UPPERCASE
  GIVE: "GIVE"
  LOOK AT: "LOOK AT"
  WITH: "with"


=============================================================================
#  MAIN MENU - ACTIVE WITH F2
#=============================================================================
menus:
  FILE_TITLE: "FILE"
  SAVE_CMD: "SAVE"
  LOAD_CMD: "LOAD"
  HELP_TITLE: "HELP"
  NO_OPT: "NO"
  GAME_HELP_OPT: "GAME HELP"
```

9. CONFIG FILE

```
# ========================================
#  MAIN CONFIGURATION - EDIT HERE!
# ========================================
CONFIG = {
    # --- INTERNAL DIMENSIONS (Virtual Canvas) ---
    # This is the "real" resolution of your graphics. If doing pixel
    # art, use low values (e.g., 320x240).
    # If using HD graphics, use high values (1920x1080).
    "GAME_WIDTH": 800,
    "GAME_HEIGHT": 638, # Total height including the interface (Verbs + Inventory).

    # --- WINDOW DIMENSIONS (What the user sees) ---
    # The engine will stretch the game to fill this window while maintaining
    # the aspect ratio.
    # Recommendation: Start with a comfortable working resolution, like 1280x720.
    "WINDOW_WIDTH": 960,
    "WINDOW_HEIGHT": 766,

    # --- GAMEPLAY ---
    "PLAYER_SPEED": 3.5,        # Character movement speed (Pixels per frame).
    "TEXTBOX_HEIGHT": 40,       # Height of the black bar where "Look at Bell" text appears.
    "VERB_MENU_HEIGHT": 128,    # Height of the verb and inventory panel (bottom part).
    "BOTTOM_MARGIN": 20,        # A small extra margin at the bottom of the window.

    # --- CAMERA (Scroll) ---
    # Controls how smoothly the camera follows the player.
    # Low values (1.0) = Slow/heavy camera. High values (10.0) = Instant camera.
    "CAMERA_SMOOTHING": 5.0,

    # --- PATHFINDING (Movement Intelligence) ---
    # Defines how the character calculates the route to avoid obstacles.
    # "EUCLIDEAN": The most accurate and natural (allows any angle).
```

```
# "MANHATTAN": Retro grid style (moves only in a cross pattern).
# "DIAGONAL":  Allows diagonals but prefers straight lines.
"PATHFINDING_TYPE": "EUCLIDEAN",

# Size of the invisible navigation grid.
# 5 = Very accurate (the character passes through small gaps).
# 20 = Less accurate, better performance (style of very old games).
"PATHFINDING_GRID_SIZE": 10,

# --- NARRATIVE STYLE ---
# Defines where text appears when someone speaks.
# "LUCAS": Floating colored text above the character's head (Monkey Island).
# "SIERRA": Centered text box with portrait (King's Quest / Larry).
# "SUBTITLE": Fixed text at the bottom of the screen (Movies).
"NARRATION_STYLE": "LUCAS",

# --- SYSTEM ---
"ENABLE_SOUND": True,       # True = Sound on. False = Muted (useful for fast development).

# --- DEVELOPMENT MODES (DEBUG) ---
"DEBUG_MODE": False,        # If True, shows collision boxes and error console.
"SHOW_HINTS_ONLY": False,   # If True, only shows object names (as player aid).
"SHOW_WALKABLE_MASK": False,# (F4 Key) Shows walkable areas in red. Useful for artists.

# --- CURSOR ---
# "MODERN": Changes icon based on action (Eye, Hand, Feet...). Requires images in /cursor.
# "CLASSIC": Uses a simple cross (+) for everything, classic SCUMM style.
"CURSOR_STYLE": "CLASSIC",

"DOUBLE_CLICK_MS": 500,     # Time in milliseconds to detect double click (fast exit).
```

```
    # --- "IDLE" ANIMATION ---
    # Time in seconds the character must stay still to perform their "special"
    # animation (e.g., scratching, checking watch).
    "IDLE_COOL_THRESHOLD": 10.0,

    # --- QUICK START (Developers only) ---
    # Scene ID to skip menu and intro. Useful for testing a specific room.
    # Set to "None" to play normally from title.
    "DEV_START_SCENE": "None", # Example: "DARK_ROOM"
}

# ========================================
#  PLAYER CONFIGURATION
# ========================================
PLAYER_CONFIG = {
    "NAME": "Gilo",                # Internal name (for logs and debug).
    "ASSET_PREFIX": "gilo",        # IMPORTANT! Image file prefix.

    # If you put "gilo", the engine will look for
    # "gilo_wd.gif", "gilo_talk.gif", etc.
    "TEXT_COLOR": (255, 255, 255),  # Text color when this character speaks (RGB).
    "CHAR_ID": "Gilo"              # ID that connects with the definition below (CHAR_DEFS).
}

# ========================================
#  CHARACTER DEFINITIONS (Spritesheets)
# ========================================
# Here you register EVERY character with complex
# animations (Player or NPCs).
CHAR_DEFS = {
    "Gilo": {
```

```
        "prefix": "gilo",   # File prefix.
        "width": 163,       # Width of EACH individual frame inside the spritesheet.
        "height": 300,      # Height of EACH frame.
        "base_scale": 0.3,  # Initial scale (0.3 means it will be seen at 30% of original size).

        # Animation map: Defines how many frames each action has.
        # If your "gilo_wd.gif" (walk down) file has 6 drawings, put 6 here.
        "frames": {
            "walk_down": 6, "walk_left": 6, "walk_right": 6, "walk_up": 6,
            "talk_down": 6, "talk_left": 6, "talk_right": 6, "give": 6,
            "idle": 1,      "push": 1,     "pull": 1,       "pick": 1,
            "open": 1,      "close": 1,     "cool": 6,
        }
    },
    # You can add as many characters as you want copying the block above:
    "Bart": {
        "prefix": "bart",
        "width": 163,
        "height": 300,
        "base_scale": 0.3,
        "frames": {
            "walk_down": 6, "walk_left": 6, "walk_right": 6, "walk_up": 6,
            # ... rest of animations ...
        }
    },
}

# ========================================
#  GAME CREDITS
# ========================================
# Text that appears when pressing "Credits" in the main menu.
```

```
CREDITS_TEXT = """
================================
    YOUR ADVENTURE GAME
================================
    Created by: Your Name
================================
    PyCAPGE ENGINE
================================
    Base code: Garba
    ...
"""

# ========================================
#  TEXT CONFIGURATION (Fonts)
# ========================================
# Path to your .ttf font file. If it doesn't
# exist, it will use Arial by default.
UI_FONT_PATH = os.path.join("fonts", "ui_font.ttf")

TEXT_CONFIG = {
    # Speaking speeds (seconds per letter).
    # Lower number = Faster.
    "SPEED_SLOW": 0.15,
    "SPEED_MEDIUM": 0.08, # Default speed.
    "SPEED_FAST": 0.04,

    # Font sizes for dialogs.
    "SIZE_SMALL": 18,
    "SIZE_MEDIUM": 20,
    "SIZE_LARGE": 32,
```

```
# Initial active configuration
"CURRENT_SPEED": "SPEED_MEDIUM",
"CURRENT_SIZE":  "SIZE_MEDIUM",
"FONT_NAME":     UI_FONT_PATH,
"OUTLINE_WIDTH": 2     # Thickness of the black border around letters (for readability).
```

Here is the Technical Reference Manual for config.py of the PyCAPGE engine.

The config.py file is the static core of PyCAPGE. This is where the global constants that determine the behavior of the engine, the resolution, the physics of movement, the aesthetics of the interface, and the definition of the character sprites are defined.

Golden Rule: This file is designed to be modified. However, Python syntax (dictionaries, lists, Boolean/integer/floating-point data types) must be strictly adhered to.

9.1 Dimensions and Resolution

PyCAPGE separates the "Logical Resolution" (the pixel art canvas) from the "Physical Resolution" (the Windows window).

GAME_WIDTH / GAME_HEIGHT

- *Description: Defines the internal resolution of the game. It is the size of the canvas where backgrounds and characters are drawn before being scaled.*
- *Recommendation: For a "Retro HD" or crisp pixel art style, use 800x450 (16:9) or 640x360. If you are looking for a classic VGA (SCUMM) style, use 320x200 or 320x240..*
- *Impact:* It directly affects performance. Lower resolutions consume fewer resources and facilitate pixel art..

WINDOW_WIDTH / WINDOW_HEIGHT

- *Description: Defines the initial size of the window opened by the operating system..*
- *Operation: The engine automatically scales what happens in GAME_WIDTH to fill WINDOW_WIDTH, applying smoothing filters (smoothscale) or pixel-perfect filters according to the internal configuration..*
- *Usage: It is recommended to set a modern standard resolution (e.g., 1280x720 or 1920x1080) so that the game looks good on current monitors when it starts up..*

For the demo, I used a resolution of 800 * 638. This allows me to draw the visible screen with a resolution of 800*450,

which is a standard 16:9. The rest is taken up by the text area and verbs. Later, to view it, I increased it by 20%.

My advice is to do it the other way around: start with very high resolutions, and then use a smaller one in the window to make the game.

That way, when scaling, if someone has a large screen, it will look sharper..

Start with very high resolutions, and then use a smaller one in windowed mode to play the game..

9.2 Pathfinding System

```
# --- PATHFINDING ---

# "EUCLIDEAN": More accurate, natural movement (uses square root).
# "MANHATTAN": Faster, ideal for city-like grids (no diagonals).
# "DIAGONAL":  (Chebyshev) Fast, allows diagonals but less accurate than Euclidean.
"PATHFINDING_TYPE": "EUCLIDEAN",
"PATHFINDING_GRID_SIZE": 10, # 5 for high precision, 20 for performance/retro
```

Control how characters navigate the stage while avoiding obstacles.

- *"EUCLIDEAN": More natural and organic movement. Calculates actual distances (straight line). Recommended for modern adventures.*
- *"MANHATTAN": Grid-based movement (only up, down, left, right). Useful for very strict retro games or cities with straight streets.*
- *"DIAGONAL" (Chebyshev): Allows diagonals but calculates the cost in a simplified way. A middle ground in performance.*

9.3 Narrative and visual Style

```
# NARRATION SYSTEM
# "LUCAS": Floating text above the character's head.
# "SIERRA": Text inside a centered box (like a comic or narrator).
# "SUBTITLE": Fixed text at the bottom of the screen.
"NARRATION_STYLE": "LUCAS",
```

- "LUCAS": Floating text above the character's head. The color of the text depends on the character's config. (Monkey Island style).
- "SIERRA": Text in a centered box or portrait (although the current implementation centers the text).
- "SUBTITLE": Fixed text at the bottom of the screen, movie theater style..

		
Lucas	Subtitle	Sierra

9.4 Cursor Style

- *Options:*
  - CLASSIC: Draws a simple cross (+) using code. Ideal for prototyping or minimalist styles.
  - MODERN: Uses animated .gif or .png graphics that change depending on the action (eye to look, hand to grab, etc.).

    Requires the files to exist in the cursor/ folder.

9.5 Development Tools

```
"ENABLE_SOUND": True,
"DEBUG_MODE": False,
"SHOW_HINTS_ONLY": False,
"SHOW_WALKABLE_MASK": False,    # VARIABLE TO SHOW WALKABLE MASK F4---
```

F3 – DEBUG MODE (coord x,y)

F1. GAME HELP

F4 - WALKABLE

9.6 Player Settings:

Define who the protagonist is when starting the game..

```
# ========================================
#  DEFAULT PLAYER CONFIGURATION
# ========================================
PLAYER_CONFIG = {
    "NAME": "Gilo",          # Name to display in texts/dialogues (Options: Bart, Gilo, Indy, Garba)
    "ASSET_PREFIX": "gilo",  # File prefix (e.g., "player_walk.gif") (Options: bart, indy, garba)
    "TEXT_COLOR": (255, 255, 255), # white
    "CHAR_ID": "Gilo"
}
```

9.7 Character Definition:

This is the most complex and powerful section for artists. Here you define how the engine should "cut out" and animate the spritesheets.

Each character is a key in the dictionary (e.g., "Gilo," "Bart").

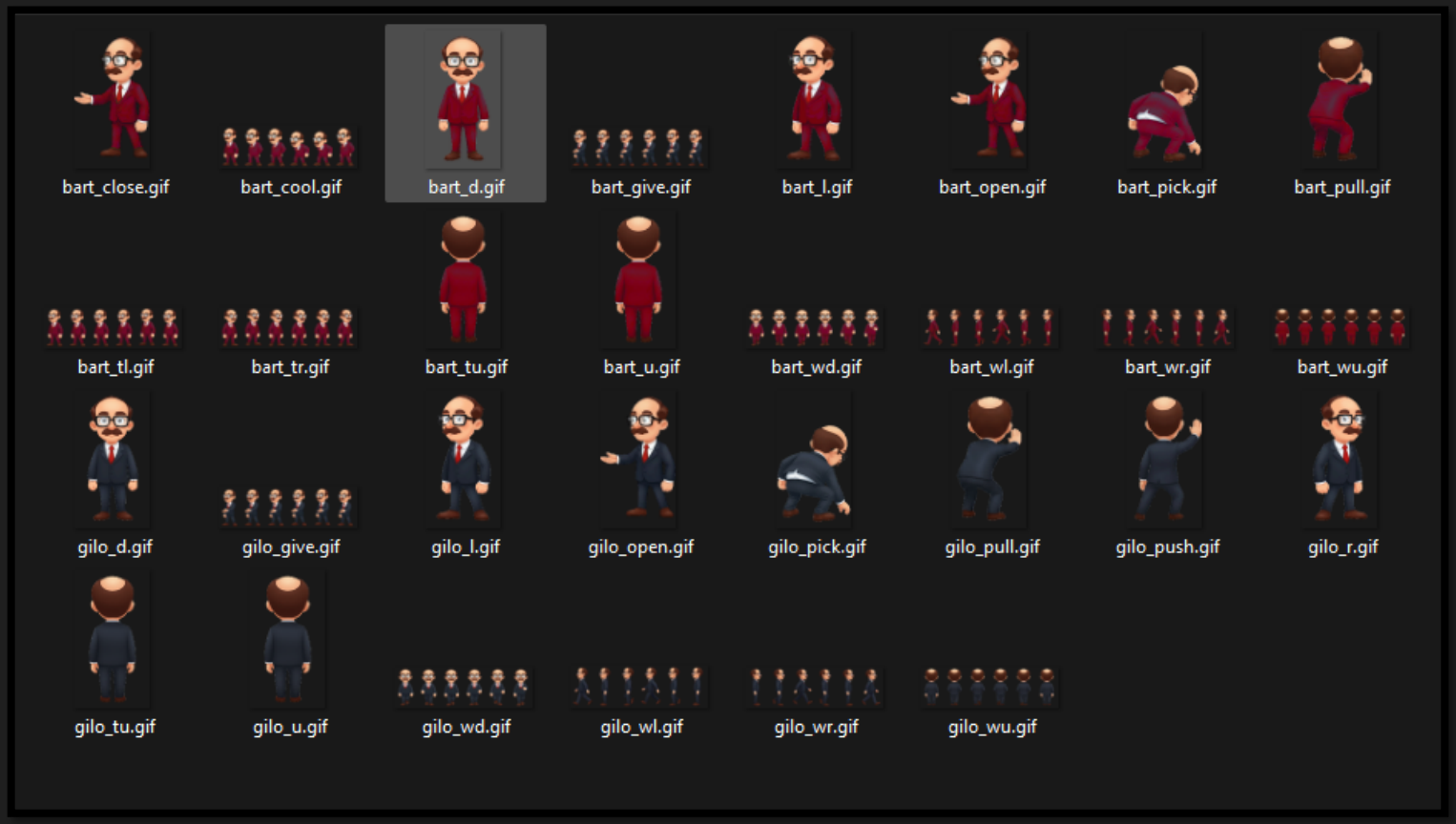
9.8 Character Parameters:

prefix: The base string of the file name. The engine will search for files with the pattern: prefix_action.gif.

- *o Example:* If prefix is "indy," it will search for indy_wd.gif (walk down), indy_talk_left.gif, etc.

Key in frames	File Suffix	Action
"walk_down"	_wd.gif	Walking down (front)
"walk_left"	_wl.gif	Walk left
"walk_right"	_wr.gif	Walk straight ahead
"walk_up"	_wu.gif	Walking up (back)
"talk_down"	_td.gif	Speak facing forward
"talk_left"	_tl.gif	Talk left profile
"talk_right"	_tr.gif	Speak right profile
"give"	_give.gif	Action of giving an object
"idle"	_d.gif / _l.gif...	Rest state (usually 1 frame)
"push"	_push.gif	Push
		etc

- Dimensions of a single frame within the sprite sheet. All frames in an animation must be the same size.
- base_scale: Initial scale of the sprite. 1.0 is original size. 0.3 reduces to 30%. This is then multiplied by the depth scale of the scene.
- frames: Dictionary indicating how many frames each specific animation has.
- It is best illustrated with some examples: In the ITEMS directory, we have the NPCs



- • We have this file: gilo_wd.gif

```
"Gilo": {
    "prefix": "gilo",
    "width": 163,       # Frame width
    "height": 300,      # Frame height
    "base_scale": 0.3,  # Starting scale
    "frames": {
        "walk_down": 6, "walk_left": 6, "walk_right": 6, "walk_up": 6,
        "talk_down": 6, "talk_left": 6, "talk_right": 6, "give": 6,
        "idle": 1,      "push": 1,      "pull": 1,      "pick": 1,
        "open": 1,      "close": 1,     "cool": 6,
    }
},
```

We see that we have defined it with a width of 163 pixels and a height of 300 pixels each, and that the animation consists of 6 items..

Then, to insert it into the default game, we have assigned it a very high scale: 0.3.

This means that we made the drawing very large for the final resolution.

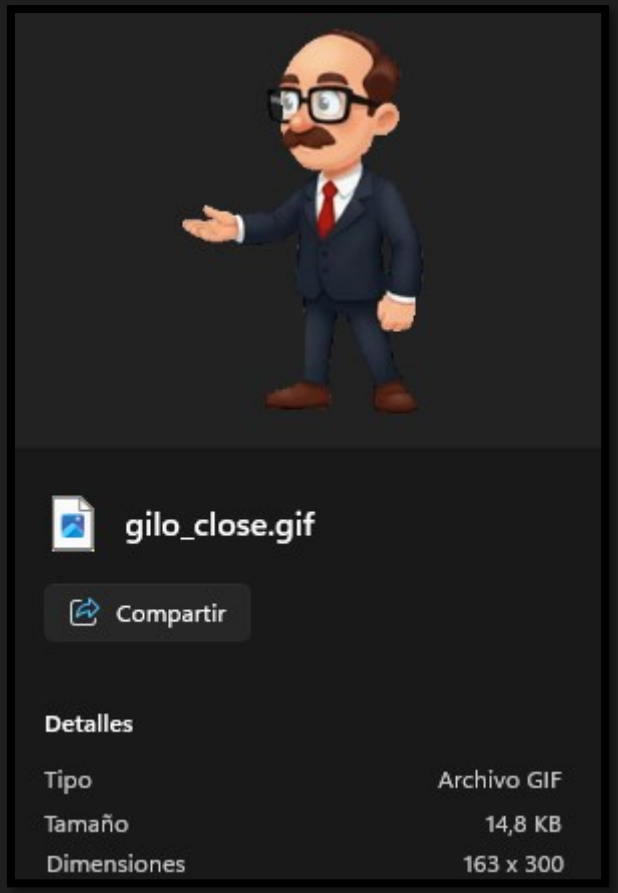
Therefore, as indicated, it is better to start with high resolutions

This way, we define more NPCs.

Taking into account the assigned name and the prefix for the graphics.

Another example, in this case, of an item without animation

With the name: gilo_close.gif –> An image. 163 x 300 - Close":1



10. INTRO SCENE

A special start or presentation scene has been created that users can use or create their own with a cutscene or however they prefer.

But to make things easier, this intro with slides may be useful.

The variable self.playlist is a list of slides. Each element within the brackets [...] is a dictionary { ... } representing a screen or "slide"

in the sequence..

```
# ========================================
#  SLIDE CONFIGURATION (INTRO)
# ========================================

def start_intro(self):
    # 1. LOAD TEXTS
    # Retrieves the cinematic text dictionary loaded from the current YAML.
    # This ensures that if you switch languages, 'cine_texts' contains the
    # correct translations.
    cine_texts = self.get_texts()



    # 2. PLAYLIST DEFINITION (SLIDE SEQUENCE)
    # A list of dictionaries is defined. Each dictionary is a "slide" or scene.
    # The manager (IntroManager) will show them in sequential order.
    self.playlist = [

        # --- SLIDE 1: Start ---
        {
```

```
    "image": "avda_paz.jpg",    # Background image (/backgrounds folder).

    # Narrative text to display.
    # (YAML es.yaml -> cinematics -> INTRO_1)
    # -> "Year 2001. The city seemed quiet..."
    "text": cine_texts["INTRO_1"],

    "effect": "none"            # No zoom effect, static image.
},

# --- SLIDE 2: The Conflict ---
{
    "image": "ayun.jpg",        # Switching to the City Hall image.

    # (YAML es.yaml -> cinematics -> INTRO_2)
    # -> "But something dark was brewing at City Hall."
    "text": cine_texts["INTRO_2"],

    "effect": "none"            # Static image.
},

# --- SLIDE 3: The Call ---
{
    "image": "darkness-room.jpg",    # Dark image.

    # (YAML es.yaml -> cinematics -> INTRO_3)
    # -> "Only a hero (or two) could stop it."
    "text": cine_texts["INTRO_3"],

    "effect": "none"
```

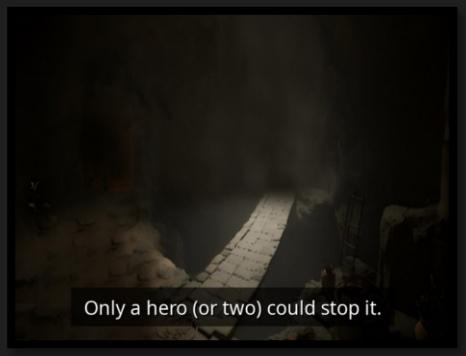
```
    },

    # --- SLIDE 4: Logo and Music ---
    {
        "image": "logo_pycapge.png",     # Engine or game logo.

        "text": "",                # Empty string: Shows no text, only image.

        # Start music synchronized with this slide.
        # Plays "sintonia3.ogg" from the /snd folder.
        "music": "sintonia3.ogg",
        "music_loops": 0,            # 0 = Plays only once (no loop).

        # --- VISUAL EFFECT (ZOOM) ---
        "effect": "zoom_out",       # The image starts large and zooms out.

        "zoom_intensity": 1.0,           # High intensity: Starts at double size (200% -> 100%).

        # Custom duration for this slide (overwrites default of 6.0s).
        "duration": 10.0,        # Stays on screen for 10 seconds.
    }
]
# 3. INITIALIZATION
self.current_index = 0             # Pointer to the start of the list.
self.active = True                 # Activates the intro rendering loop.
self.load_current_slide()          # Loads the first image into memory.
        # Changes the global game state so main.py knows it must draw the Intro.

self.set_state(GameState.INTRO)
```

11. ENDING SCENE

```
# ========================================
#  SLIDE CONFIGURATION (ENDING)
# ========================================

def start_ending(self):
    # 1. LOAD TEXTS
    # Just like in the intro, we retrieve the translated texts.
    cine_texts = self.get_texts()

    # 2. FINAL PLAYLIST DEFINITION
    self.playlist = [

        # --- SLIDE 1: Resolution ---
        {
            "image": "darkness-room.jpg",    # Reusing the dark background.

            # (YAML es.yaml -> cinematics -> ENDING_1)
            # -> "And so, after unearthing the secret..."
            "text": cine_texts["ENDING_1"],

            "effect": "none",

            # Triumphal or credits music that starts here.
            "music": "sintonia1.ogg"
        },

        # --- SLIDE 2: The Calm ---
        {
            "image": "avda_paz.jpg",
```

```
        # Return to the initial scene (circular closure).

        # (YAML es.yaml -> cinematics -> ENDING_2)
        # -> "The city returned to normal."
        "text": cine_texts["ENDING_2"],

        # Subtle zoom out effect.
        "effect": "zoom_out",
        "zoom_intensity": 0.2           # Low intensity: Starts at 120% and reduces to 100%.
    },

    # --- SLIDE 3: Credits / Acknowledgments ---
    {
        "image": "logo_pycapge.png",    # Final Logo.

        # (YAML es.yaml -> cinematics -> THANKS)
        # -> "THANKS FOR PLAYING"
        "text": cine_texts["THANKS"],

        "effect": "zoom_out",
        "zoom_intensity": 0.5,          # Medium zoom.
        "duration": 6.0,                # Minimum display time.

        # --- WAIT FOR INPUT ---
        # This "wait_for_input": True property is critical.
        # It means the game will NOT return to the main menu automatically
        # when the time runs out. It will stay frozen on this image
        # until the player presses a key or clicks.
        "wait_for_input": True
    }
]
```

```
# 3. INITIALIZATION
self.current_index = 0
self.active = True

# Changes global state to ENDING. main.py will stop processing game logic
# and will switch to calling ending_manager.draw() and ending_manager.update().
self.set_state(GameState.ENDING)
self.load_current_slide()
```



Slides

12. SCENE 1 – PEACE AVENUE

```
# ============================================================================
#  SCENE 1: Peace Avenue
# ============================================================================

# 1. SCENE OBJECT CREATION
# ----------------------------------------------------------------------------
# Scene(INTERNAL_ID, VISIBLE_NAME, BACKGROUND, MASK, SCALING, Y_RANGE, TRANSITION)
s1 = Scene(
    "AVDA_PAZ",
# Unique ID to reference this scene in
# code. Save games, etc.
    SCENE_NAMES["AVDA_PAZ"],    # Translated name [YAML: "Peace Avenue"].
    "avda_paz.jpg",
# Background image (backgrounds/ folder).
    "avda_paz_bm.jpg",
# Walkable mask (white = walkable, black = wall).
    scale_range=(0.4, 2.2),     # PC scaling: 0.4 at the back (far), 2.2 at the front (near).
    y_range=(230, 400),
# Vertical limits where that scaling
# applies (Y pixels).
    transition_type=TRANSITION_FADE # Visual effect upon entering (Fade to black).
)

# 2. ON ENTER EVENT
# ----------------------------------------------------------------------------
# Automatically executed every time the player enters this
# room. Always with lambdas.
# Here we launch a floating welcome text.
s1.on_enter = lambda: game_play_event(
```

```
texto=OBJ_DESCS["SCENE1_INTRO_MSG"], # [YAML: "THIS IS DEMO TEXT FOR THE USER"]
    text_time=8,        # The text will last 8 seconds on screen.
    pos=(100, 100)      # Position (X, Y) where the floating text will appear.
)

# 3. HOTSPOT 1: THE BELL (Fixed interactive object)
# --------------------------------------------------------------------------------
s1.add_hotspot_data(

name="campana",          # Internal object ID.

image_file="campana.png",# Object image (objects/ folder).
    x=480, y=200,
  # Screen position (feet/base coordinate).
    walk_to=(480, 330),
# Where the character will stand to interact with this.
# Use whenever it's far away.
    label_id="BELL",         # Name visible on mouse hover [YAML: "Bell"].
    hint_message="BELL_HINT",# Hint if help is pressed [YAML: "Use the hammer here!"].
    scale=0.1,
    # Size of the bell image (10% of original).
    primary_verb="LOOK AT",  # Default verb on right click.

    # Dictionary of reactions to verbs:
    actions={
        "LOOK AT": "BELL_LOOK", # [YAML: "It's a really big bell."]
        "OPEN":    "BELL_OPEN", # [YAML: "It's solid bronze."]
        "PUSH":    "BELL_PUSH", # [YAML: "Ding!"]
        "PULL":    "BELL_PULL", # [YAML: "Doooooooonng! What a clapper."]
```

```
        # --- COMPLEX INTERACTION: USE OBJECT ON OBJECT ---
        # The key is formed like this: USE + INVENTORY_ITEM_ID + _ON_ + HOTSPOT_ID
        "USE_MARTILLO_ON_CAMPANA": lambda: game_play_event(

    texto=OBJ_DESCS["BELL_MAGIC_USE"], # [YAML: "The sound resonates twice as loud with magic!"]



    # List of sounds to play in
    # sequence or mix.

    play_sound=["church-bell.ogg", "medal"],



        # Activates a global variable
        # to remember this was already done.

    flag="puzzle_campana_resuelto",



    # Deletes the hammer from inventory after using it.

    delete_item="martillo"

),
        }
    )
```

```
# 4. HOTSPOT 2: THE HAMMER (Collectable object)
# ------------------------------------------------------------------------------
s1.add_hotspot_data(

name="martillo",

image_file="martillo.png",
    x=450, y=315,
    label_id="HAMMER",            # [YAML: "Hammer"]
    scale=0.08,
        # Image scaling.

    # 'flag_name' is CRUCIAL for pickable objects. Global variable of your game.
    # If 'martillo_recogido' is True in variables.py, this object will NOT be drawn.
    flag_name="martillo_recogido",

    description="HAMMER_LOOK",    # Generic description.
    primary_verb="LOOK AT",       # Right click = Look at.



actions={

    "LOOK AT": "HAMMER_LOOK", # [YAML: "Carpenter's hammer."]



    # Having 'flag_name' and the verb "PICK UP",
    # the engine automatically knows:
        # 1. To put it in the inventory.
        # 2. To mark the flag as True.
```

```
        # 3. To make the object disappear from the scene.
        # 4. To show the associated text: [YAML: "Okay, I'll take it."] In
        # the inventory I can put another image or the same one, as preferred.
        "PICK UP": "HAMMER_PICK"

}
)

# 5. EXITS
# ------------------------------------------------------------------------------
# Invisible zone that changes scenes.
# add_exit(x, y, width, height, TARGET_SCENE_ID, spawn_x, spawn_y)

# Right exit towards the Town Hall
s1.add_exit(
    x=750, y=0, w=50, h=GAME_AREA_HEIGHT, # Invisible rectangle on the right of the screen.
    target_scene="TOWN_HALL",             # ID of the scene to load (defined in s2).
    spawn_x=50, spawn_y=400               # Where the character will appear in the new scene.
)

# Left exit towards the Panoramic View
s1.add_exit(
    x=0, y=0, w=50, h=GAME_AREA_HEIGHT,   # Invisible rectangle on the left.
    target_scene="PANORAMIC",             # ID of the scene to load (defined in s3).
    spawn_x=1500, spawn_y=400             # Spawn coordinates in the Panoramic View.
)
 # 6. FINAL REGISTRATION
# ------------------------------------------------------------------------------
# We add the configured scene to the game manager so it works.
scene_manager.add_scene(s1)
```

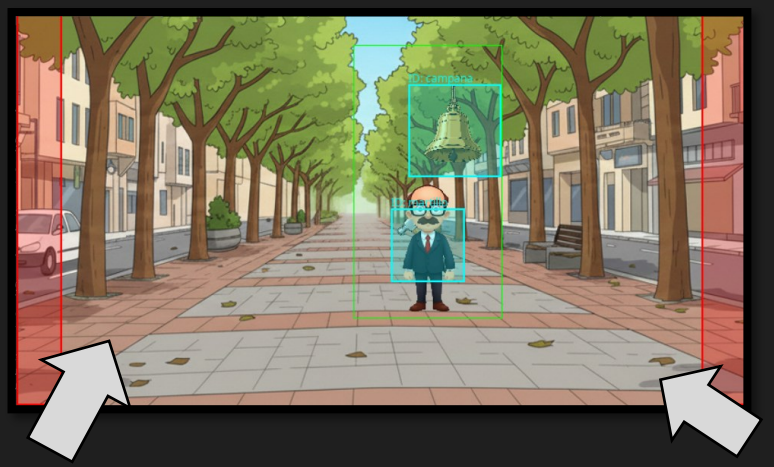
Principal Image avda_paz.jpg (png, webp, etc)

Walkable zone avda_paz_bm.jpg (png, webp, etc)

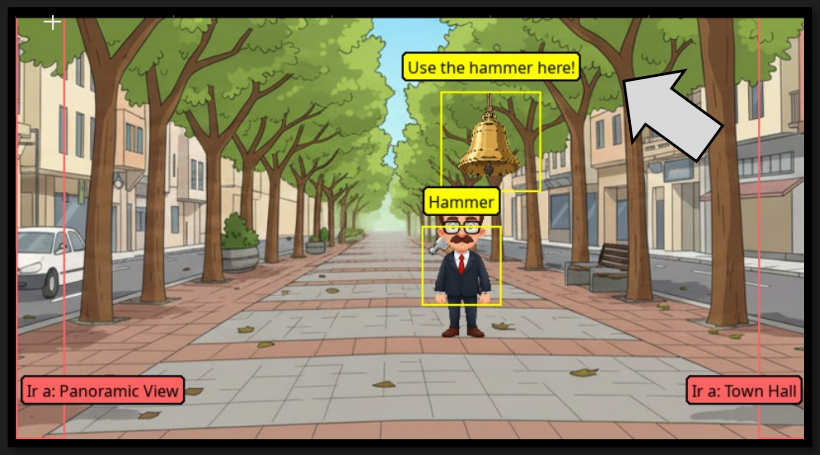
Perspective – Scale Range

Perspective - Scale Range

Demo text



Doble Click to Run



Game Help and Game Clues

Default Verb

13. SCENE 2 – TOWN HALL

```
# ========================================
#  SCENE 2: Town Hall
# ========================================

# 1. SCENE DEFINITION WITH AMBIENCE
# ------------------------------------------------------------------------------
s2 = Scene(

"TOWN_HALL",               # Internal ID.

SCENE_NAMES["TOWN_HALL"],  # Name [YAML: "Town Hall"].
    "ayun.jpg",
# Background image.
    "ayun_bm.jpg",
# Walkable mask.
    scale_range=(0.1, 1.8),    # Very aggressive scaling (very small at back, big near).
    y_range=(230, 400),
# Vertical scaling range.

    # Visual effect: The scene slides in from the left.
    transition_type=TRANSITION_SLIDE_LEFT,



# Specific step sound for this room
# (wooden floor).
    step_sound_key="step_wood",
```

```
# Lighting file: An image that
# defines what color/tint
    # the character has depending on where they step (e.g., orange near fire, gray
    # for a dark alley, etc.).
    lightmap_file="ayun_light.jpg"
)

# 2. INTRODUCTION CUTSCENE
# ------------------------------------------------------------------------------
# We define a local function for the automatic intro.
def intro_ayuntamiento():
    # We check a variable (flag) so this ONLY happens the first time.
    if not GAME_STATE.get("intro_ayuntamiento_vista", False):



    # List of instructions for the engine's
    # "Movie Director".
        acciones = [

        {"type": "WAIT", "seconds": 1.0},           # Wait 1 sec.

        {"type": "MOVE", "x": 400, "y": 450},       # Move PC to these coordinates.

{"type": "FACE", "dir": "camera"},       # Make them face the
# screen/camera.

{"type": "WAIT", "seconds": 0.5},           # Dramatic pause.
```

```
        # Automatic dialogues
        # (SAY). The player cannot skip them (unless special key).

# CINE_TEXTS comes from 'cinematics' in the
# YAML.

{"type": "SAY", "text": CINE_TEXTS["AYUN_1"], "time": 2.5}, # [YAML: "I was here 25 years ago..."]

{"type": "SAY", "text": CINE_TEXTS["AYUN_2"], "time": 2.0}, # [YAML: "...nothing has changed."]



{"type": "WAIT", "seconds": 0.5},



# IMPORTANT: At the end, we mark the flag
# as True to never repeat this again.

{"type": "FUNC", "func": lambda: GAME_STATE.update({"intro_ayuntamiento_vista": True})}
        ]
        # Execute the sequence.
        cutscene_manager.start_cutscene(acciones)

# Assign this function to the on enter event.
s2.on_enter = intro_ayuntamiento

# 3. EXITS
# ------------------------------------------------------------------------------
# Left -> Returns to the Avenue.
```

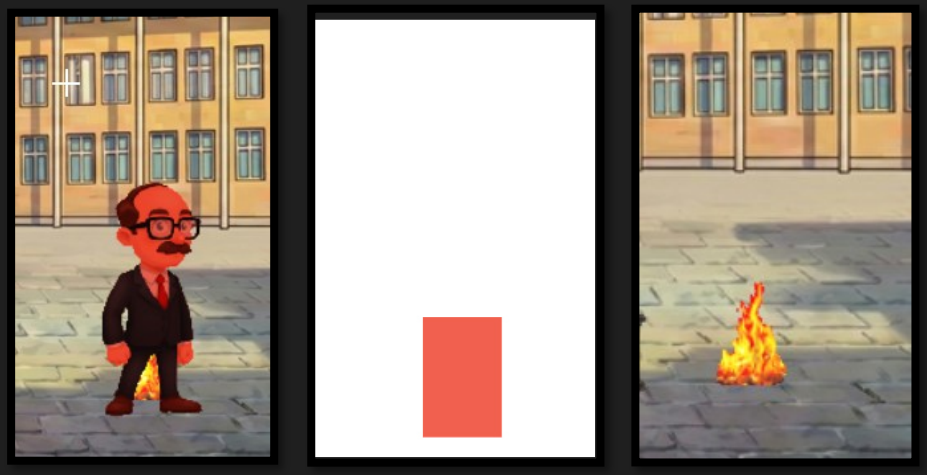
```
s2.add_exit(x=0, y=0, w=50, h=GAME_AREA_HEIGHT, target_scene="AVDA_PAZ", spawn_x=750, spawn_y=400)
# Right -> Goes to the Dark Room.
s2.add_exit(x=750, y=0, w=50, h=GAME_AREA_HEIGHT, target_scene="DARK_ROOM", spawn_x=50, spawn_y=400)

# 4. ADVANCED HOTSPOT: COMBINATION PUZZLE
# ------------------------------------------------------------------------------
# PRO Tip: We build the dictionary key dynamically.
# The engine looks for: USE_[ITEM_NAME]_[ON]_[HOTSPOT_NAME]
# Using ITEM_NAMES['STONE'] ensures we use the correct name for the
# stone object.
key_use_stone = f"USE_{ITEM_NAMES['STONE'].upper()}_ON_VENTANA_VECINA"

s2.add_hotspot_data(

name="ventana_vecina",

x=60, y=90, width=100, height=100, # Invisible zone (no image, only
# coordinates).
    label_id="WINDOW_OLD",            # [YAML:
# "Old Window"]
    walk_to=(110, 380),
        # Where the PC stands
# to look.
    primary_verb="LOOK AT",

actions={

    "LOOK AT": "WINDOW_LOOK",
# [YAML: "It's Mrs. Fletcher's window."]
        "OPEN":    "WINDOW_OPEN",      # [YAML: "It's stuck, it's been painted over for years."]
```

```
        # Reaction when using the STONE on the WINDOW.
        key_use_stone: OBJ_DESCS["WINDOW_STONE"] # [YAML: "Better not break anything
# today."]

}
)

# 5. STANDARD HOTSPOT: PICK UP OBJECT
# ---------------------------------------------------------------------------
s2.add_hotspot_data(

name="farol",

image_file="farol.png",

x=600, y=420, scale=0.1,
    label_id="LANTERN_OLD",        # [YAML: "An
# old lantern"]
    flag_name="farol_recogido",    # If True, it is not drawn (you already picked it up).
    primary_verb="PICK UP",

actions={

    "LOOK AT": "LANTERN_LOOK", # [YAML: "A
# lantern from Aliexpress."]

    "PICK UP": "LANTERN_PICK", # [YAML: "I'll take it."]
    }
)

# 6. ANIMATED HOTSPOT: MACHINE
```

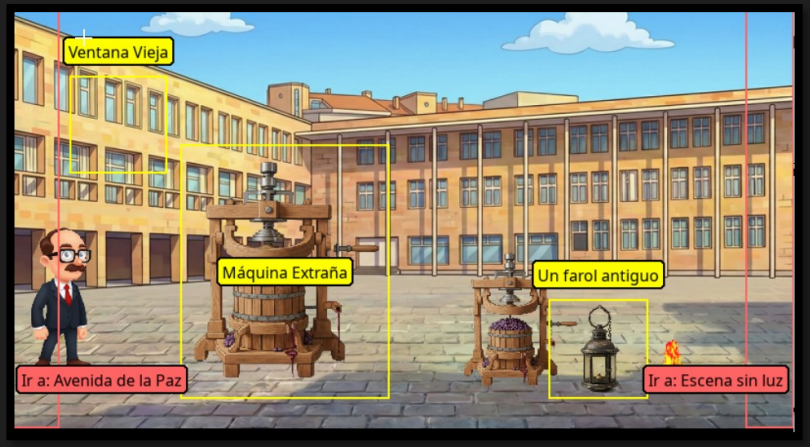
```
# ------------------------------------------------------------------------------
# This object has animated graphics and collision (it is solid).
s2.add_hotspot_data(

name="maquina_demo",

image_file="animation_demo.png",

num_frames=5, anim_speed=200,    # Sprite Sheet
# configuration.

x=280, y=420, scale=0.65,
    solid=True,
            # Pathfinding will avoid it.
    label_id="MACHINE_STRANGE",      # [YAML: "Strange
# Machine"]
    primary_verb="USE",

actions={

    "LOOK AT": "MACHINE_LOOK",   # [YAML:
# "It looks like complex machinery."]



    # On USE, calls a helper function
    # that plays the animation once.
        "USE":  lambda: play_object_animation("maquina_demo", OBJ_DESCS["MACHINE_USE"]), # [YAML:
# "It works!"]

    "PUSH": lambda: play_object_animation("maquina_demo", OBJ_DESCS["MACHINE_PUSH"]) # [YAML:
```

```
# "I give it a little push..."]
    }
)

# 7. AMBIENT ELEMENT (DECORATION): BONFIRE
# ------------------------------------------------------------------------------
# 'add_ambient' adds visual elements that are not necessarily
# interactive (clickable),
# although here we added actions to serve as a hybrid example.
s2.add_ambient(
image_file="hoguera.gif",
x=675, y=420,
    num_frames=50,   # Long animation.
    anim_speed=50,   # Fast speed (fire).
    scale=1.0,
    layer="back",    # "back" = Behind the character /
# "front" = In front (covering the PC).
    solid=True,      # You cannot walk on fire.
    label_id="BONFIRE", # [YAML: "Bonfire"]
    actions={
        "LOOK AT": "BONFIRE_LOOK", # [YAML: "I get red if I get too
# close..."]
    }
)
# We add the scene to the engine
scene_manager.add_scene(s2)
```

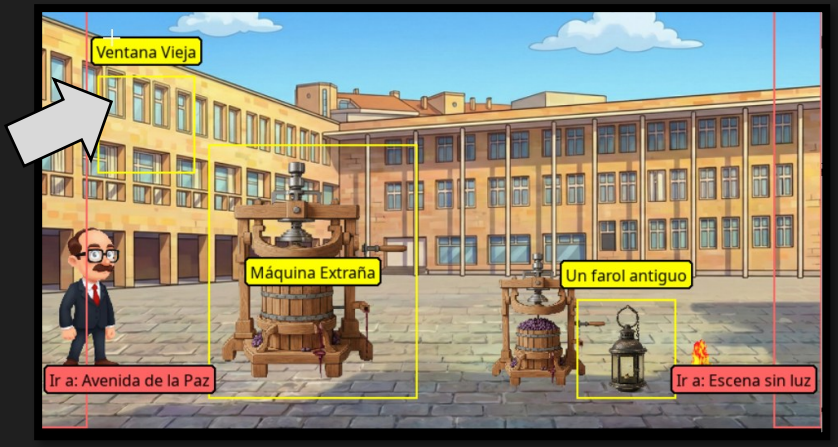
CutScene -Automatic – ESC to exit



changing the palette with a background



Standars Hotspots , and animated objects



Special hotpspot

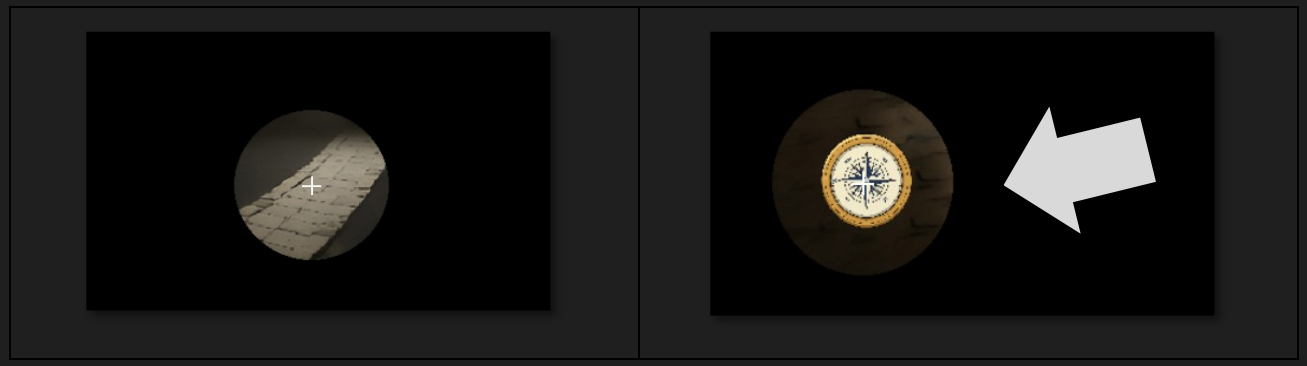
14. SCENE 4 – DARK ROOM

```
# ========================================
#   # --- SCENE 4: DARK SCREEN ---
# ========================================

# 1. SCENE DEFINITION
# Here we instantiate the Scene class with its basic properties.
s4 = Scene(
    "DARK_ROOM",
    # INTERNAL ID: Must be unique. Used to travel here (target_scene).
    SCENE_NAMES["DARK_ROOM"],      # VISIBLE NAME: Comes from YAML ("Dark scene").
    "darkness-room.jpg",
    # BACKGROUND IMAGE: File in the /backgrounds folder.
    "darkness-room_bm.jpg",
    # WALK MASK: Defines where one can step (white/black).

    scale_range=(1.75, 2.2),
    # CHARACTER SCALE: (Minimum, Maximum).

    # The character will look 1.75x at the back and 2.2x at the front.
    y_range=(325, 400),
    # Y RANGE: Vertical pixels where scaling is applied.

    # (From Y=325 to Y=400).
    # --- LIGHT CONFIGURATION (Special for this scene) ---
    is_dark=True,                  # ACTIVATE DARKNESS: The screen will be covered in black.
    light_flag="farol_recogido",   # LIGHT CONDITION: Variable name in GAME_STATE.


    # If GAME_STATE["farol_recogido"] is True, there will be light.
```

```
    light_radius=70,               # LIGHT RADIUS: Size of the transparent circle around the mouse.
    transition_type=TRANSITION_FADE # TRANSITION: Visual effect upon entering (Fade to black).
)
# 2. ON ENTER EVENT DEFINITION (on_enter)
# This function will execute AUTOMATICALLY every time the player enters this room.
def enter_dark_room():
    # We verify if we do NOT have the lantern collected
    if not GAME_STATE.get("farol_recogido", False):
    # If there is no light, we launch a complaint message from the character.
        # YAML: DARK_ROOM_ENTER: "I can't see a thing! I need light."
       game_play_event(texto=OBJ_DESCS["DARK_ROOM_ENTER"], text_time=3.0)
# We assign the created function to the scene's on_enter event.
s4.on_enter = enter_dark_room

# 3. MAP SYSTEM CONFIGURATION (Fast Travel)
# We define a list of destinations. Each line is a point on the map.
# Format: (SCENE_ID, MAP_X, MAP_Y, SPAWN_X, SPAWN_Y, ICON)
map_destinations = [
    # The dark room itself (marked on the map at x=250, y=350). When traveling you appear at 400, 390.
    ("DARK_ROOM", 250, 350, 400, 390, "pin.png"),
    # The main street.
    ("AVDA_PAZ",  400, 500, 400, 390, "pin.png"),
    # The town hall.
    ("TOWN_HALL", 200, 300, 435, 420, "pin.png"),
    # The panoramic view.
    ("PANORAMIC", 500, 360, 975, 400, "pin.png"),
    # The Parallax scene (Tech demo).
    ("PARALLAX",  375, 250, 590, 370, "pin.png")
]
# 4. INTERACTIVE OBJECT: THE MAP (Lying on the ground)
s4.add_hotspot_data(
```

```
name="mapa1",                  # Unique object ID in this scene.
image_file="mapa1.png",        # Object image (in /items or /objects folder).
x=250, y=400, scale=0.1,       # Position and size.

label_id="MAP",                # TRANSLATABLE LABEL: Looks for "MAP" in YAML.

# YAML items -> MAP: "Old Map"
flag_name="tengo_mapa",        # EXISTENCE FLAG:

# If "tengo_mapa" is True, this object is NOT DRAWN in the scene

# (because we already picked it up). If False, it appears on the ground.

primary_verb="PICK UP",        # DEFAULT VERB: On right-click or hover.
solid=True,
     # SOLID: The player will have to walk around it, cannot walk through it.
actions={
    # LOOK Action
    # YAML descriptions -> MAP_LOOK: "A tourist map."
    "LOOK AT": "MAP_LOOK",
# PICK UP Action
# YAML descriptions -> MAP_PICK: "I'll take it."
    # Upon picking it up, automatically sets "tengo_mapa"=True and adds the item to the inventory.
    "PICK UP": "MAP_PICK",
    # USE Action (Advanced conditional logic with lambda)
    # Here we ask: Do I have the map in my possession?
    "USE": lambda: load_and_open_map(map_destinations, "mapa1.jpg")
            if GAME_STATE.get("tengo_mapa")  # IF I HAVE IT: Opens the map interface.
    else game_play_event(texto=OBJ_DESCS["MAP_FAIL"]), # IF I DON'T HAVE IT (it's on the ground):

                                # YAML: "I can't use it lying there..."
```

```
            # OPEN Action (Same logic as use)
            "OPEN": lambda: load_and_open_map(map_destinations, "mapa1.jpg")

if GAME_STATE.get("tengo_mapa")
else game_play_event(texto=OBJ_DESCS["MAP_FAIL"])
  }
  )
  # 5. EXITS
  # We define an invisible zone that leads us to another scene.
  s4.add_exit(
      x=0, y=0, w=50, h=GAME_AREA_HEIGHT, # Activation rectangle (Left edge of the screen).
      target_scene="TOWN_HALL",           # Target: Scene 2 ID (Town Hall).
      spawn_x=750, spawn_y=400            # Where the character appears in the new scene.
  )
  # 6. FINAL REGISTRATION
  # We add the configured scene to the game manager.
  scene_manager.add_scene(s4)
```



15. SCENE 3 – PANORAMIC STREET

```
# ========================================
#   SCENE 3: Panoramic (Scroll and Advanced Mechanics)
# ========================================

# 1. BASIC SCENE DEFINITION
# ------------------------------------------------------
# Scene(INTERNAL_ID, VISIBLE_NAME, BG_IMAGE, WALK_MASK, ...)
s3 = Scene(
    "PANORAMIC",
    # Unique ID used by the code to refer to this scene.
    SCENE_NAMES["PANORAMIC"],      # Visible name (seen when saving game). [YAML: "Panoramic"]
    "panoramica.jpg",
    # Background image (must be wide to allow scrolling).
    "panoramica_bm.jpg",
    # Walkability mask (white = walkable, black = wall).

    # scale_range: (Top scale, Bottom scale).
    # Here (1.9, 2.1) means the character looks very big (zoom),
    # ideal for close-ups. 1.0 is original size.
    scale_range=(1.9, 2.1),

    # y_range: Vertical limits (pixels) where the character can walk.
    y_range=(325, 400),

    # Specific step sound for this scene (e.g., rug or dirt).
    step_sound_key="step_rug",
```

```
  # Transition type upon entering (slide up).
      transition_type=TRANSITION_SLIDE_UP
  )

  # 2. "ON ENTER" EVENT
  # ------------------------------------------------------------
  # This lambda function executes automatically right after loading the scene.
  s3.on_enter = lambda: (
      # Plays background music, with a 5-second fade and volume at 50%.
      play_scene_music("sintonia2.ogg", duration_s=5, volume=0.5),



  # Shows an introductory floating text.
      # [YAML: "This is an example of a screen with horizontal SCROLL"]
      game_play_event(texto=OBJ_DESCS["SCENE3_INTRO_MSG"], pos=(400, 150), text_time=4.0)
  )

  # 3. INTERACTIVE OBJECT WITH STATES: TRASH CAN
  # ------------------------------------------------------------
  # This object has two visual states (closed/open) controlled by frames.
  s3.add_hotspot_data(

  name="basura",
# Internal object ID.

  image_file="trashcan.png",     # Sprite sheet (must contain frames side by side).
      num_frames=2,                  # The sprite has 2 images (Frame 0: Closed, Frame 1: Open).
      anim_speed=0,                  # Speed 0 so it doesn't animate itself (we control it).
      x=800, y=355,
         # Initial position in the world.
```

```
    label_id="TRASHCAN",           # Name on mouse hover. [YAML: "Trash Can"]
    primary_verb="LOOK AT",        # Default verb (right click).
    scale=0.15,
         # Object size relative to its original image.
    solid=True,
         # The character cannot walk through it (collision).

    # walk_to: Where the character stops to interact (X, Y).
    walk_to=(875, 365),


    # facing: Where the character looks when interacting ("left" = looks left).
    facing="left",

    actions={
        # LOOK Action: Shows description.
        # [YAML: "It's full of waste."]
        "LOOK AT": "TRASH_LOOK",

        # OPEN Action: Executes a complex function.
        "OPEN": lambda: (

# Changes the object graphic to frame 1 (lid open).

change_state_object("basura", 1),

# Shows text and the character performs "open" animation.

# [YAML: "Yuck! What a stench."]

        game_play_event(texto=OBJ_DESCS["TRASH_OPEN"], anim="open")
```

```
    ),

        # CLOSE Action: Returns to frame 0 (lid closed).
        "CLOSE": lambda: (

        change_state_object("basura", 0),

        # [YAML: "Better closed."]

        game_play_event(texto=OBJ_DESCS["TRASH_CLOSE"], anim="close")

    ),
    }
)

# 4. DIALOGUE SYSTEM (NPC GARBA)
# ------------------------------------------------------
# We define the conversation tree in a function to load texts on the fly.
def get_garba_dialogue_tree():

return {

    "start": {  # Initial node

        "options": [

            {

        # Conditional option: Only appears if you have the flashlight with batteries.
```

```
        # [YAML: "Look! I managed to put batteries in the flashlight."]

        "text":
DIALOGUE_TEXTS.get("GARBA_START_OPT1", "GARBA_START_OPT1"),



                # NPC response.

        # [YAML: "Great! I knew you were handy."]

        "response":
DIALOGUE_TEXTS.get("GARBA_START_ANS1", "GARBA_START_ANS1"),



                "condition": "linterna_con_pilas_recogida", # Flag required in GAME_STATE.

        "next": "tengo_la_linterna_con_pilas"        # Jumps to another node.

    },

    {

        # Standard option.

        # [YAML: "Hi Garba, what are you doing here?"]

        "text":
DIALOGUE_TEXTS.get("GARBA_START_OPT2", "GARBA_START_OPT2"),
```

```
                    "response": DIALOGUE_TEXTS.get("GARBA_START_ANS2", "GARBA_START_ANS2"),
                    "next": "quiero_una_linterna_con_pilas"
        },
        {
            # Exit option ("EXIT" closes the dialogue).
            "text":
    DIALOGUE_TEXTS.get("GARBA_START_OPT4", "GARBA_START_OPT4"),
                    "response": DIALOGUE_TEXTS.get("GARBA_START_ANS4", "GARBA_START_ANS4"),
                    "next": "EXIT"
        }
    ]
},
            # ... (Definition of other nodes like "I_want_a_flashlight..." etc.)
 "quiero_una_linterna_con_pilas": {
     # ... node options ...
 },
 "tengo_la_linterna_con_pilas": {
```

```
    # ... node options ...

}
        }

    # NPC definition in the scene
    s3.add_hotspot_data(

    name="garba",

    image_file="garba_tr.gif",      # Animated GIF (idle).

    num_frames=6, anim_speed=300,   # Animation configuration.
        x=1250, y=360,
        scale=0.65,
        text_color=(255, 255, 0),       # Text color when this NPC speaks (Yellow).
        label_id="NPC_GARBA",           # [YAML: "Garba"]

    primary_verb="TALK TO",

    actions={
            # On TALK: Starts the dialogue system with the tree defined above.
            "TALK TO": lambda: dialogue_system.start_dialogue(

            get_garba_dialogue_tree(),

            "start",

            npc_ref=s3.hotspots.get_hotspot_by_name("garba") # Reference so mouth moves.
```

```
    ),

    "LOOK AT": "NPC_GARBA_LOOK", # [YAML: "What are you looking at? Do I have monkeys on my face?"]

        # Special Action: GIVE OBJECT (Give)
        # The key is formed automatically: GIVE + ITEM_ID + ON + HOTSPOT_ID
        "GIVE_LINTERNA_CON_PILAS_ON_GARBA":

    lambda: game_play_event(

        texto=OBJ_DESCS["GARBA_GIVE_FLASHLIGHT"], # [YAML: "Here Garba, you keep it."]

  delete_item="linterna_con_pilas",
# Removes the item from inventory.

  play_sound="medal"
       # Success sound.
        )
    }
  )

  # 5. CRAFTING: FLASHLIGHT AND BATTERIES
  # ------------------------------------------------------------
  # Object A: Flashlight without batteries
  s3.add_hotspot_data(

  name="linterna_sin_pilas",

  image_file="linterna_sin_pilas.png",
      x=950, y=370, scale=0.085,
      label_id="FLASHLIGHT_EMPTY",            # [YAML: "A flashlight without batteries"]
```

```
    flag_name="linterna_sin_pilas_recogida", # Global variable to know if we already picked it up.
    primary_verb="PICK UP",

actions={

    "PICK UP": "FLASHLIGHT_EMPTY_PICK",  # [YAML: "I'll take it."]



    # Combination: USE BATTERIES on FLASHLIGHT (Inverse)
        "USE_PILAS_PARA_LINTERNA_ON_LINTERNA_SIN_PILAS": lambda: crafting(

"linterna_sin_pilas",       # Ingredient 1 (ID).

"pilas_para_linterna",      # Ingredient 2 (ID).

"linterna_con_pilas",       # ID of the resulting New Object.

"linterna_con_pilas.png",   # Image of the new object.

"linterna_con_pilas_recogida" # Flag to activate.
        )
    }
)

# Object B: Batteries (Symmetric logic to allow A on B and B on A)
s3.add_hotspot_data(

name="pilas_para_linterna",

image_file="pilas_linterna.png",
```

```
x=1050, y=370, scale=0.08,

label_id="BATTERIES",

flag_name="pilas_linterna_recogidas",
    primary_verb="PICK UP",

actions={

    "PICK UP": "BATTERIES_PICK",

        # Combination: USE FLASHLIGHT on BATTERIES
        "USE_LINTERNA_SIN_PILAS_ON_PILAS_PARA_LINTERNA": lambda: crafting(

"linterna_sin_pilas",

"pilas_para_linterna",

"linterna_con_pilas",

"linterna_con_pilas.png",

"linterna_con_pilas_recogida"
        )
    }
)

# Result Object: Complete Flashlight (Defined off-screen)
# It is necessary to define it so the inventory knows what description to show.
# I draw it in a very far place so it doesn't disturb.
```

```
s3.add_hotspot_data(

name="linterna_con_pilas",

image_file="linterna_con_pilas.png",
    x=-1000, y=-1000,
# Negative coordinates so it is not seen in the scene. But it must exist defined.
    label_id="FLASHLIGHT_FULL", # [YAML: "A Lit Flashlight"]
    primary_verb="USE",

actions={

    "LOOK AT": "FLASHLIGHT_FULL_LOOK", # [YAML: "Now it works, shining brightly."]

    "USE":  "FLASHLIGHT_FULL_USE",
    }
)

# 6. SPECIFIC LOGIC PER CHARACTER: THE SHOVEL
# ------------------------------------------------------------
s3.add_hotspot_data(

name="pala",

image_file="objects_pala.png",

x=570, y=360, scale=0.1,
    label_id="SHOVEL",            # [YAML: "Heavy Shovel"]
    flag_name="pala_recogida",

primary_verb="PICK UP",
```

```
actions={
        # Conditional logic on PICK UP:
        "PICK UP": lambda: (

# CASE 1: If we are Bart (The strong one)

(

            inventory.add_item("pala", ITEM_NAMES["SHOVEL"], "objects_pala.png"),

            GAME_STATE.update({"pala_recogida": True}),

            # We visually remove the object from the scene:

    [h.kill() for h in scene_manager.get_current_scene().hotspots.hotspots if h.name == "pala"],

            # Success message [YAML: "Humpf! It weighs a ton..."]

            game_play_event(texto=OBJ_DESCS["SHOVEL_BART_PICK"], play_sound="medal")

        )

        if PLAYER_CONFIG["CHAR_ID"] == "Bart"



        # CASE 2: If we are Gilo (The weak one)

# Fail message [YAML: "It weighs too much. I need Bart."]
```

```
        else game_play_event(texto=OBJ_DESCS["SHOVEL_GILO_FAIL"], speaker=player)

    ),



    # Different description depending on who looks

    "LOOK AT": lambda: game_play_event(texto=OBJ_DESCS["SHOVEL_LOOK_BART"])

            if PLAYER_CONFIG["CHAR_ID"] == "Bart"

            else game_play_event(texto=OBJ_DESCS["SHOVEL_LOOK_GILO"])

}
)

# 7. AMBIANCE: MOVING OBJECT (BIRD)
# ----------------------------------------------------------
s3.add_ambient(

image_file="crow8.gif",

num_frames=8, anim_speed=120, # 8 sprite sheet frames, step each frame every 120 millis.

x=-50, y=100,               # I give it a negative X so it starts outside the screen.
    scale=0.3,
    layer="back",          # Draws behind the character.
    solid=False,
# Does not collide.
```

```
    # Automatic movement:
    move_to=(1600, 150),   # Final destination (off-screen to the right).
    move_speed=100,        # Flight speed.
    loop_move=True,        # Upon reaching the end, returns to start (loop).

    label_id="BIRD",       # [YAML: "Big Bird"]

actions={

    "LOOK AT": "BIRD_LOOK" # [YAML: "Oh, look, birdies!"]

}
)

# 8. CHARACTER SWAP (PLAYABLE NPCs)
# ------------------------------------------------------
# NPC BART (Used to switch control to him)
s3.add_hotspot_data(

name="Bart",

image_file="bart_d.gif",

x=180, y=360, scale=0.6,
    label_id="NPC_BART",         # [YAML: "Bart"]
    flag_name="controlando_bart",# If True, this NPC hides (because you are controlling him).
    primary_verb="TALK TO",

solid=True,

actions={
```

```
    "LOOK AT": "NPC_BART_LOOK",

    "TALK TO": "NPC_BART_TALK",

    # On USE: Switches player control to "Bart".
        "USE": lambda: change_player_active("Bart"),

    "PUSH": "NPC_BART_PUSH"
    }
)

# NPC GILO (Same to return to Gilo)
s3.add_hotspot_data(

name="Gilo",

image_file="gilo_d.gif",

x=180, y=360, scale=0.6,
    label_id="NPC_GILO",         # [YAML: "Gilo"]
    flag_name="controlando_gilo",

primary_verb="TALK TO",

solid=True,

actions={

    "LOOK AT": "NPC_GILO_LOOK",
```

```
    "TALK TO": "NPC_GILO_TALK",

    "USE": lambda: change_player_active("Gilo"),

    "PUSH": "NPC_GILO_PUSH"
    }
)

# 9. EXITS AND FINALIZATION
# ------------------------------------------------------
# On exit event: Stop music.
s3.on_exit = lambda: stop_scene_music()

# Define exit zone (Invisible rectangle at the far right).
s3.add_exit(

x=1550, y=0, w=50, h=GAME_AREA_HEIGHT, # Detection rectangle.

target_scene="AVDA_PAZ",            # Target scene ID.
    spawn_x=50, spawn_y=400               # Where the player spawns in the new scene.
)

# Important! Add the scene to the global manager.
scene_manager.add_scene(s3)
```

16. SCENE 5 – PARALLAX EXAMPLE

```
 # ========================================
 #  # --- SCENE 5: PARALLAX EXAMPLE
 # ========================================

 # 1. SCENE INSTANTIATION
 # Creates the Scene object. This scene is special because it uses the
 # layer system (Parallax). Using webp as an example.
 s5 = Scene(
     "PARALLAX",
      # Internal Scene ID (used for scene switching and logic).
     SCENE_NAMES["PARALLAX"],         # Name to show in UI/Debug.


    # (YAML es.yaml) -> "Parallax Demo"

     "parallax_middle.webp",
 # Base background image (Fallback). If the parallax system
 # fails, this one loads.
     "parallax_middle_bm.webp",      # Navigability mask (Bitmask). Defines where the
 # PC can walk (white=yes, black=no).

     # --- PARALLAX CONFIGURATION (Visual Layers) ---
     parallax_paths=[
         "parallax_far.webp",
# Layer 0: Background (Sky/Distant mountains). Painted first.
         "parallax_middle.webp",      # Layer 1: Mid-plane (Ground/Buildings). Where the PC walks.
         "parallax_near.webp"
# Layer 2: Foreground (Bushes/Fog). Covers the character.
     ],
```

```
    # Speed factors for depth effect when moving the camera.
    parallax_factors=[
        0.0,      # Factor 0.0: Layer 0 does not move with the camera (fixed or auto-scroll).
        1.0,      # Factor 1.0: Layer 1 moves synchronized with the camera (base).
        2.0       # Factor 2.0: Layer 2 moves at double speed (proximity effect).
    ],

    # Auto-scroll configuration independent of the player.
    # (Layer index, X Speed).
    auto_scroll_config=(0, -15.0),   # Layer 0 (Sky) moves by itself to the
    # left at speed 15. With 0 = automatic. I can put another

    # Character scaling (Zoom).
    scale_range=(1.5, 2),
   # Min scale (at back) 1.5x, Max scale (at front) 2.0x.
    y_range=(350, 500)
    # Y range in pixels where that scaling applies (from y=350 to y=500).
)

# 2. "ON_ENTER" LOGIC (Upon entering the scene)
# A lambda function is defined that executes multiple actions when loading the screen.
s5.on_enter = lambda: (
    # A) Play ambient music.
    # File: 'fores_bird_pymapge.ogg', no limit duration (0), volume 70%.
    play_scene_music("fores_bird_pymapge.ogg", duration_s=0, volume=0.7),



# B) Show introductory narrative text.
    # Looks up the key in the descriptions dictionary loaded from YAML.
```

```
    # (YAML es.yaml -> descriptions) -> "THIS IS THE FINAL SCREEN"
    game_play_event(texto=OBJ_DESCS["SCENE5_INTRO_MSG"], text_time=8.0)
)

# 3. DYNAMIC KEY PREPARATION (For Puzzles)
# This is best practice: Construct the action name using the real object name.
# If you translate "SHOVEL" to "Pala" or "Shovel" in YAML, the code still works.
# The engine expects the format: USE + ITEM_NAME + _ON_ + HOTSPOT_NAME

# (YAML es.yaml -> items -> SHOVEL) -> "Heavy Shovel"
# Expected string result: "USE_PALA PESADA_ON_MARCA_FINAL" (The engine manages spaces internally)
key_dig_shovel = f"USE_{ITEM_NAMES['SHOVEL'].upper()}_ON_MARCA_FINAL"

# 4. HOTSPOT DEFINITION (Interactive objects)
# We define the "X" on the ground marking the end of the game.
s5.add_hotspot_data(
    name="marca_final",              # Unique hotspot ID in this scene.
    image_file="x_the_end.png",      # Graphic sprite of the X.
    x=620, y=400,
         # Coordinates (x, y) where it is drawn.
    scale=0.4,
            # Sprite scaling to 40%.

    # Label visible on mouseover.
    # (YAML es.yaml -> items -> MARK_X) -> "Mark on the ground"
    label_id="MARK_X",

    primary_verb="LOOK AT",          # Default verb if double-clicked or right-clicked.

    # Action Dictionary (Verb -> Reaction)
    actions={
```

```
        # LOOK Action:
        # (YAML es.yaml -> descriptions) -> "A giant X painted on the ground. Like in pirate maps."
        "LOOK AT": "MARK_LOOK",

        # USE Action (Generic, no object):
        # (YAML es.yaml -> descriptions) -> "I need something to dig."
        "USE":     "MARK_USE_FAIL",



    # PUSH Action:

    # (YAML es.yaml -> descriptions) -> "The ground is too hard."
        "PUSH":    "MARK_PUSH_FAIL",


        # --- PUZZLE RESOLUTION (Use Shovel on Mark) ---
        # The specific key for combining objects is defined.
        # Note: Here "USE_PALA_ON_MARCA_FINAL" is hardcoded instead of using the variable `key_dig_shovel` created above.
        # This assumes the internal ID of the inventory item is "pala".
        "USE_PALA_ON_MARCA_FINAL": lambda: (

# Step 1: Show initial success text.

# (YAML es.yaml -> descriptions) -> "There is something here! I'm going to dig..."

        game_play_event(texto=OBJ_DESCS["MARK_DIG_START"], text_time=2.0),



        # Step 2: Start a Cutscene (Controlled cinematic)
```

```
cutscene_manager.start_cutscene([

    # Event 1: The character says a phrase.

    # (YAML es.yaml -> descriptions) -> "Crack! I hit something solid."

    {"type": "SAY", "text": OBJ_DESCS["MARK_HIT_OBJ"], "time": 4.0},



            # Event 2: Execute game end function.

    # Calls the ending manager to switch to ENDING state.

    {"type": "FUNC", "func": ending_manager.start_ending}

        ])
        )
    }
)

# 5. "ON_EXIT" LOGIC (Upon exiting the scene)
# Important to stop sounds that shouldn't play on the next screen (credits or menu).
s5.on_exit = lambda: stop_scene_music()

# 6. FINAL REGISTRATION
# The configured scene is added to the global manager so the engine can load it.
scene_manager.add_scene(s5)
```

17. A BIT OF HISTORY

Back in the summer of ’84, while Orwell was coming to life on TV screens (and Van Halen was crushing it on the radio), I discovered black magic in someone else’s house. An older kid—half friend, half spiritual guide—took us to a well-to-do family’s home. You know the type: they had a pool and pristine walls without a single scuff mark. And there, in the living room, the spell was cast: a ZX Spectrum hooked up to the TV. A rubber keyboard with the power to summon aliens onto the screen. They played; I watched. Maybe they let me press a key, shoot a couple of pixels, or move the paddle in *Pong*, but mostly I just sat there mesmerized, like I was seeing fire for the first time. *(Excerpt from my book, ISBN available... lol)*

By 1985, I had my own rig: an Amstrad CPC664 with a green phosphor monitor. The floppy disks that came with the system included some very basic games written in BASIC: *Teletenis*, *Plaga*, *Pulga*, and *Animal Vegetal Mineral*. The cool thing about those games was that you had immediate access to the code. I was never great at playing them; I was always more interested in popping the hood to see how the thing worked. I spent that entire first year bathing in green light.

The first magazines featuring "POKE" codes and a book on BASIC programming my dad bought me made the winter in our mountain town a lot more bearable. Soon after, I got my hands on my first non-graphic games: text adventures. They were Spanish-made, too. They didn't grab my attention much as games, but I definitely tore them apart to see how they ticked. I was 12 years old; I wanted a little action.

Shortly after that, *it* arrived. The great Spanish masterpiece: *La Abadía del Crimen* (The Abbey of Crime). Beating *La Abadía* is an achievement I still list on my resume. It wasn’t a graphic adventure like the ones that blew up later, but it marked a turning point in the genre.

A couple of years later, in high school, I had an English teacher who was a bit ahead of his time. He’d request the budding computer lab and, on Fridays, let us play *King’s Quest* on PCs with those 5¼-inch floppies. Now *that* was something else. Color screens and characters you could actually interact with. It was basically the Yankee version of *La Abadía*.

The big-time graphic adventures arrived just a few years later. The "Holy Trinity": *Monkey Island*, *Indiana Jones and the Fate of Atlantis*, and *Day of the Tentacle*. You can’t just pick one. Those were the first games that truly hooked me from start to finish.

During my first years in college, I remember how it was in the dorms—whenever someone discovered something cool, they’d sprint to the other floors to spread the word. Yeah, we should

have been studying, but... come on. That same year, I heard about the release of *Igor: Objective Uikokahonia*, the first graphic adventure developed in Spain (1994). That was the ultimate challenge. If Pendulo Studios could build that marvel, I wanted to do it too.

We were already studying ASM and C++ in our classes, so I took a crack at it, but I didn't have the resources, the tools, or the know-how. With some help, I managed to make something resembling a text adventure like *Animal Vegetal Mineral*—the first game I'd loaded onto my CPC nearly a decade prior—but not much else. Honestly, I'd gotten better results with BASIC as a kid.

A couple of years later, I installed a library for C++ that promised miracles: Allegro. I spent a few months tinkering with it, but I just didn't have the time. I had to study. And that was the end of my second attempt at making a graphic adventure: "Harder than Rock." It had a title and everything, and the code is still rotting on a hard drive somewhere, actually doing stuff.

It wasn't until my senior year in 2000 that I got serious. By then we had the internet; you could ask questions in forums, and there was a lot of advanced info out there. So, working in bits and pieces over a year, I churned out 25,000 lines of code and finished my first Point-and-Click graphic adventure. It ran on the Java Virtual Machine, making it cross-platform and playable online. I think that was a pretty big achievement for the time. It was big enough that, 20 years later, the local press picked up the story and even interviewed me. Back in the day, I tried to give it away to the press, but they didn't have a clue what I was talking about. I remember winning a local contest with it, though. I used the prize money to buy my first Fender Stratocaster.

Since then, I've kept programming as a hobby and done a bit of professional work, though not so much these days.

That old engine eventually stopped working with OS updates, but I still had that itch to scratch—I wanted to build a new system that was more compatible. I'm not really a Python guy, but right now it offers the most options, and let's be real: with AI, bugs get fixed in minutes.

So, over the last few months, I've developed this engine that clones all the features of those classic Point-and-Click games from the early 90s. I've tried to replicate the behaviors faithfully, and I hope I've created a functional engine so I can make my own adventures—and so other users can make theirs.

Thanks, folks.

ENGINE & CODE:

- garba eduardogarbayo.com
- zainder.com Programmers
- riojawebs.com Graphics

SPECIAL THANKS:

- Python Community
- Pygame Developers
- LucasArts (For inspiration)
- Chir (Indy Java MAGE)

DONATIONS:

- If you want to support the development of PyCAPGE, you can donate at: https://www.zainder.com/donations/