Nice — I'll design a 4-week, **step-by-step** plan for a *simple Python maze game*, plus a **full, detailed flowchart for Week 1** (both the developer task flow and the runtime/game loop flow). I'll also give you a complete, working **Week 1 starter code** (Pygame) you can run right away and the exact files you should have at the end of Week 1.

Overview — game & approach

We'll build a small tile-based maze game in Python using **Pygame** (graphical, easy to learn).

High level features across 4 weeks:

- **Week 1:** Setup + playable prototype — grid rendering, keyboard movement, collision with walls, one goal, restart/win. (Fully done here.)
- **Week 2:** Procedural maze generation (e.g., recursive backtracker), HUD (timer/moves), level loading.
- **Week 3:** Enemies/patrols or hazards, collectibles, simple AI/pathfinding, score saving.
- **Week 4:** Polish and package — title/menu, sound, assets, packaging with pyinstaller, playtesting & bugfixes.

# 4-Week Plan — detailed, step-by-step

Week 1 —

Playable prototype (deliverable: main.py, README.md, assets/ folder)

Goal: By the end of Week 1 you have a playable tile maze where the player can move with arrow keys (or WASD), cannot walk through walls, and can reach a goal to win.

Daily steps (Day 1 → Day 7):

- **Day 1 — Environment & project skeleton**

    - Install Python (3.8+ recommended).
    - pip install pygame
    - Create project folder:
    - Verify pygame import in a quick script.
- **Day 2 — Window + game loop skeleton**
    - Create main.py with Pygame init, window, main loop and quit handling.
    - Print a simple “Hello Pygame” rectangle to verify rendering.
- **Day 3 — Grid & rendering**
    - Create a simple hardcoded 2D grid (list of lists). Use values: 1=wall, 0=path, 2=start, 3=goal.
    - Implement draw_grid() to draw cells as squares.
- **Day 4 — Player representation & movement**
    - Represent player as a grid coordinate (r,c).
    - Add keyboard input (arrow keys/WASD) to move by 1 tile on KEYDOWN.
    - Check bounds; prevent movement into tiles where grid[r][c] == 1.
- **Day 5 — Win detection & restart**
    - Add a goal tile and check when player reaches it.
    - On win: display “You Win!” overlay and allow R to restart or Q/ESC to quit.
- **Day 6 — Basic polish & testing**
    - Add simple HUD text: level name, moves count.
    - Adjust tile size, window scale, add small margins.
    - Play through and debug collisions and win logic.

- **Day 7 — Document & commit**
    - Write a small README.md that explains how to run.
    - Save and test on another machine if possible.

Week 1 acceptance criteria

- Game opens and shows a maze grid.
- Player moves tile-by-tile using arrow keys/WASD.
- Player cannot move through walls.
- Reaching the goal triggers a “win” screen; R restarts; Q quits.
- A main.py file and README.md are present.

Week 2 — Maze generation & multiple levels

- Replace the static maze with procedural generation (recursive backtracker or Prim’s).
- Load level arrays from levels/level1.json etc.
- Add timer and move counter; store best times in scores.json.

Week 3 — Gameplay features

- Add simple enemy (patrol) using waypoints; optionally add BFS pathfinding chase.
- Add collectibles (keys, powerups) and locked doors.
- Improve UI: lives, levels, restart confirmations.

Week 4 — Polish & distribution

- Title/menu, settings (sound, controls), pause screen.
- Add sound effects/music (pygame.mixer).
- Create installer/executable: pyinstaller --onefile main.py (Windows example).
- Playtest, fix bugs, prepare a short README or gameplay instructions.

# Week 1 — Full flowchart(s)

Below are two flowcharts for Week 1:

1. **Development / Task flowchart** — what you do as a dev during Week 1.
2. **Runtime/game loop flowchart** — the logic your game will execute at runtime.

1) Development (task) flowchart — Week 1

🧩

Week 2 — Add Levels or Maze Generation

🎯

Goal

Make the game replayable — either by:

- Adding **multiple mazes (levels)**,

  or

- Automatically **generating random mazes.**

You can pick either (or both later).

🪜

Option 1: Multiple Levels (easiest)

What to do:

1. Create a list of multiple maze layouts.
2. Loop through each maze one by one.
3. When you win one level, load the next.

The code I just gave doesn't have a formal "name," but you could call it something simple like:

"Simple Maze Game"

If you want, I can also give it a slightly cooler, short name for your file, like **"EscapeMaze.py"** or **"MazeRunner.py."**

Do you want me to pick one of those?

If it's for a **coding platform or IDE**, you just need to copy the Python code I gave you into a new file in that environment. Examples:

- **VS Code** → create a new file → paste the code → save as MazeRunner.py
- **PyCharm** → same thing → new Python file → paste → save
- **Online IDE** (like Replit) → new Python project → paste → run

So the "code domain" is basically **where the Python code lives**.

Here's exactly how to run your **MazeRunner.py** game:

1. Open Terminal (Mac) or Command Prompt (Windows)

- Mac: Press **Cmd + Space**, type Terminal, press Enter
- Windows: Press **Win + R**, type cmd, press Enter

2. Navigate to the folder with your file