Title of the Project
Pac-Man Terminal Game Using C Programming

Introduction
For this project, I decided to recreate a lightweight version of Pac-Man — but only using C and the terminal. Nothing fancy. It runs right inside macOS or Linux, and the whole thing basically taught me how to handle real-time input, draw things on the screen with plain characters, and keep a game loop running without everything freezing up.

The main idea was to get some actual practice with stuff like:

- non-blocking keyboard input
- updating a 2D board in real time
- making things “move” even though you’re just redrawing characters
- and getting ghosts to behave somewhat unpredictably without being too smart

Objectives

- Build a working terminal-based game loop
- Let Pac-Man move around with W/A/S/D (without waiting for Enter every time)
- Add ghost movement that stays inside certain areas
- Keep track of a simple score and decide when the game ends
- Learn how termios and select() help with real-time input in C

System Requirements

Hardware

- A Mac or Linux system
- 4 GB RAM (honestly anything works)
- Access to a terminal

Software

- GCC or Clang
- VS Code or pretty much any editor
- A Unix-like terminal

Game Design Overview

Board Layout
The game runs on a 15×30 grid made of characters.

- # walls
- spaces for open paths
- . dots worth 10 points
- o power pellets worth 50
- P for Pac-Man
- G for ghosts

Movement
Pac-Man moves with W/A/S/D and advances every 2 cycles, while ghosts move every 3. Each ghost is limited to one quadrant and chooses a random valid direction so they don't wander all over the map.

Scoring
You get +10 for each dot and +50 for a power pellet.

- **Win**: you eat everything
- **Lose**: a ghost bumps into Pac-Man

Implementation Details

Real-Time Input
The game turns off canonical mode with termios, which basically means you don't have to hit Enter for every move. select() checks whether a key was pressed so the game doesn't pause waiting for input.

Game Loop
Each frame:

1. Clear the screen
2. Draw the board and show the score
3. Read any input
4. Move Pac-Man
5. Move the ghosts
6. Look for collisions
7. Sleep briefly (usleep())

Results
The final program actually runs pretty smoothly for a terminal game. Pac-Man and the

ghosts move independently, the score updates as you go, and the game correctly figures out when you've won or lost.

Conclusion
This was a good way to understand real-time terminal behavior — way more hands-on than just reading about it. It also shows you can make something pretty fun without using any real graphics libraries at all.

Future Improvements

- Smarter ghost AI (BFS or A*)
- Multiple levels
- Some simple animations
- Maybe sound effects
- Extra items like the original fruit bonuses
- Proper win/lose detection
- Real-time terminal gameplay

9. **Conclusion**
   This project provided practical experience with real-time terminal input, game loop design, and 2D rendering using characters. It shows that engaging games can be built without graphics libraries—just with terminal control and logic.

Possible improvements include:

- Smarter ghost AI
- Multiple levels
- Enhanced power-up features

10. **Future Enhancements**

- Implement ghost pathfinding (BFS or A* algorithms)
- Add basic animations
- Integrate sound effects using external libraries
- Include multiple maps and levels
- Introduce collectible fruit bonuses similar to the original Pac-Man