# Pac-Man 68K

Computer Structure II

Andreu Massanet Felix
Diego Malagrida González

**Abstract**

This paper presents the design and implementation of a Pac-Man-style videogame developed in MC68000 assembly using the Easy68K environment. The project focuses on low-level game architecture, grid-based movement, and enemy artificial intelligence. Ghost behavior is implemented through mode-based logic (chase, scatter, and frightened) combined with intersection-based path selection and target-driven decision making. The system integrates real-time input handling, sprite rendering, collision detection, and frame rate control within the constraints of a simulated microprocessor environment.

# 1 Game Description and Usage

This project applies the concepts learned in *Computer Structure II* to the development of a videogame. A Pac-Man-style game was implemented in the *Easy68K* environment.

The player controls Pac-Man inside a maze populated with pellets, obstacles, and enemy ghosts. Keyboard input uses `W`, `A`, `S`, `D` for movement (up, left, down, right). Pressing `P` enables a debug mode that displays each ghost's current target.

The objective is to collect all pellets without being caught by ghosts. A collision with a ghost causes the loss of one life. Standard pellets grant 10 points and *Big Pellets* grant 50 points. After consuming a *Big Pellet*, ghosts enter a temporary vulnerable state and can be eaten for 200 points each before respawning. Completing all pellets advances to the next level; losing all lives ends the game.

# 2 Implementation Overview

The codebase is split into multiple `.X68` files, each responsible for a subsystem. The main entry point `MAIN.X68` drives the execution loop and coordinates game states through `STATES.X68`.

| | |
|---|---|
| **Core** | `MAIN.X68` (main loop), `STATES.X68` (state control) |
| **Player** | `PACMAN.X68` (movement, interactions) |
| **Ghost AI** | `BLINKY.X68`, `PINKY.X68`, `INKY.X68`, `CLYDE.X68` |
| **Maze** | `MAPDATA.X68` (matrix), `MAP.X68` (rendering) |
| **Graphics** | `GFX.X68` (sprites/effects) |
| **Timing** | `FPS.X68` (frame pacing, FPS display) |
| **Score** | `SCORE.X68` |
| **Utilities** | `UTIL.X68` |
| **Globals** | `CONST.X68`, `SYSCONST.X68`, `UTLCONST.X68`, `SYSTEMVARS.X68`, `VARS.X68` |

# 3 Pac-Man Movement and Turning Logic

Pac-Man moves continuously in the current direction. Direction changes requested by the player are not applied immediately; instead, the desired direction is stored and validated at subsequent movement steps. The movement system uses predictive collision checks: both the current position and the next position are evaluated against the maze. If the desired direction becomes valid (i.e., the next cell is not blocked), the direction is updated at the earliest possible step. Otherwise, Pac-Man continues moving while the system keeps checking until the turn is feasible.

# 4 Position Mapping and Collision Computation

The maze is represented as a two-dimensional grid. Pixel-space positions are converted to grid coordinates for both Pac-Man and ghosts using:

$$X_m = \left\lfloor \frac{x}{C_s} \right\rfloor \qquad Y_m = \left\lfloor \frac{y}{C_s} \right\rfloor$$

The corresponding cell address is computed as:

$$T = MAP + (X_m + Y_m \cdot W)$$

where $C_s$ is the cell size in pixels, $W$ is the maze width in cells, and $MAP$ is the base address of the maze data.

Collision is approximated by applying this conversion to the four corners of each entity, effectively simulating a rectangular hitbox. Map cell values encode different semantics (e.g., walls, pellets, intersections, empty cells). Some cells require state updates after interactions (e.g., pellet intersections must become regular intersections after the pellet is consumed, whereas pellet-only cells become empty).

# 5 Ghost Artificial Intelligence

Ghost behavior is implemented as a mode-based system combined with intersection-based path selection. Each ghost maintains a target cell (visible in debug mode via `P`) and selects movement directions only at intersections.

## 5.1 Global Behavior Modes

Ghosts alternate between two main modes and a temporary override mode:

- **Chase**: the ghost targets Pac-Man according to its specific targeting rule.
- **Scatter**: the ghost targets a predefined corner region of the maze.
- **Frightened**: triggered when Pac-Man consumes a *Big Pellet*; ghosts attempt to flee for a limited time.

Ghosts do not necessarily start simultaneously: they leave the ghost house at different times. The system alternates between *chase* and *scatter* multiple times (three alternations) and then remains permanently in *chase*. When *frightened* is active, it overrides the current mode.

## 5.2 Ghost-Specific Targeting in Chase Mode

Each ghost computes its chase target differently:

- **Blinky (red)**: directly targets Pac-Man's current map position.
- **Pinky (pink)**: targets a position four tiles ahead of Pac-Man, based on Pac-Man's movement direction.
- **Clyde (orange)**: enforces an approximately 8-tile distance threshold; when too close, it retreats toward its home area.
- **Inky (blue)**: computes a target using both Pac-Man and Blinky; specifically, it uses the vector defined by Pac-Man and Blinky and applies a vector-doubling operation to obtain the final target.

### 5.3 Intersection-Based Path Selection

Ghosts evaluate direction changes only at cells marked as intersections in the maze matrix. At an intersection, the set of candidate directions is built by:

- discarding directions blocked by walls,
- in normal movement, discarding the reverse direction (i.e., the direction from which the ghost just arrived),
- in *frightened*, allowing reverse movement while still discarding wall directions.

For each candidate direction, the ghost predicts the resulting next cell and computes a Euclidean distance to its current target. In *chase* and *scatter*, the chosen direction is the one that **minimizes** the distance to the target. In *frightened*, the chosen direction is the one that **maximizes** the distance to Pac-Man (i.e., it selects the path that moves farthest away).

This design ensures stable movement in corridors and concentrates decision-making at controlled points, avoiding erratic behavior when encountering maze obstacles.

## 6 Additional Features

### Audio

Sound effects are implemented using `.wav` files managed through DirectX and reproduced via the corresponding system trap (e.g., pellet consumption, death, ghost capture).

### Mouse Interaction

Mouse input is used to access the instruction screen. Mouse handling is implemented through a dedicated trap separate from keyboard input.

### Dynamic FPS Control

Frame pacing is controlled through a dynamic FPS mechanism:

$$x = \frac{1000}{FPS}$$

where $x$ is the delay in milliseconds between updates for a target frame rate. The target FPS is configurable through the `FPS_TO_REACH` variable.

### High Score Persistence

The high score is stored and restored using external files. ASCII-to-number and number-to-ASCII conversion routines are used to serialize and deserialize score values.

### Sprite Rendering

Custom sprites for Pac-Man and ghosts were designed with a uniform pink background to represent transparent pixels. A Java tool converts bitmap pixel colors into hexadecimal values. Sprites are rendered pixel-by-pixel using task 82 of `TRAP #15`.