PicoMite Raycaster User Manual

Overview

The RAY command set provides a Wolfenstein 3D-style first-person renderer for the PicoMite MMBasic on RP2350. It renders textured walls, patterned floors and ceilings, full-colour billboard sprites, and includes built-in collision detection, ray casting for interaction, and a minimap overlay.

The raycaster renders directly into the current framebuffer at whatever resolution is set by the active MODE command. It reads the system HRes and VRes at render time, so it is not limited to a single resolution. Mode 2 (320x240, 4-bit RGB121 colour) is recommended for best performance. The user is responsible for creating the framebuffer and copying it to the display.

Platform: RP2350 only (PicoMite, PicoMiteVGA, HDMI).

Quick Start

```
MODE 2
CLS
DIM INTEGER map%(15)
' A simple 4x4 map: walls around the edge, open centre
map%() = 1,1,1,1, 1,0,0,1, 1,0,0,1, 1,1,1,1

FRAMEBUFFER CREATE
FRAMEBUFFER WRITE F

RAY MAP 4, 4, map%()
RAY CAMERA 2.5, 2.5, 0, 66
RAY COLOUR 12, 3, 8, 1, 1, 3

DO
  RAY RENDER
  FRAMEBUFFER COPY F, N
  k$ = INKEY$
  IF k$ = "w" THEN RAY MOVE 0.1
  IF k$ = "s" THEN RAY MOVE -0.1
  IF k$ = "a" THEN RAY TURN -5
  IF k$ = "d" THEN RAY TURN 5
  IF k$ = CHR$(27) THEN EXIT DO
LOOP

RAY CLOSE
FRAMEBUFFER CLOSE
```

Commands

RAY MAP w, h, map%()

Define the world grid.

Parameter	Description
w	Map width in cells (1-256)
h	Map height in cells (1-256)
map%()	1D integer array with w x h entries

Each array element is a wall type:
0 = empty (passable space)
1-31 = wall (rendered using the wall definition assigned by RAY DEFINE)

Every wall type from 1 to 31 has its own foreground colour, background colour, fill pattern, and an optional door flag. See RAY DEFINE to customise these. The defaults give types 1-15 green shades and types 16-31 brown/yellow shades with the door flag set.

The array is stored row by row: element y * w + x corresponds to map cell (x, y).

RAY CAMERA x!, y!, angle! [, fov!]

Place and orient the viewer camera.

Parameter	Description
x!, y!	Position in map space (floating-point, 0-based)
angle!	Heading in degrees (0 = +X/east, 90 = +Y/south)
fov!	Field of view in degrees (default 60, range 10-170)

Tip: Position the camera at x + 0.5, y + 0.5 to centre it within a map cell.

RAY COLOUR floor_fg, ceil_fg [, floor_bg, ceil_bg, floor_pat, ceil_pat]

Set the floor and ceiling appearance.

2-argument form - solid colours:

Parameter	Description
floor_fg	Floor colour (RGB121 palette index 0-15)
ceil_fg	Ceiling colour (RGB121 palette index 0-15)

6-argument form - textured with fill patterns:

Parameter	Description
floor_fg	Floor foreground colour (0-15)
ceil_fg	Ceiling foreground colour (0-15)
floor_bg	Floor background colour (0-15)
ceil_bg	Ceiling background colour (0-15)
floor_pat	Floor fill pattern index (0-31)
ceil_pat	Ceiling fill pattern index (0-31)

Pattern bits set to 1 draw in the foreground colour; bits set to 0 draw in the background colour.

Note: RAY COLOR is accepted as an alternative spelling.

RAY MOVE speed! [, strafe!]

Move the camera with built-in collision detection.

Parameter	Description
speed!	Forward speed (positive = forward, negative = backward)
strafe!	Optional. Sideways speed (positive = right, negative = left)

The engine uses a 0.25-unit bounding box around the camera. If the full move is blocked by a wall, it attempts wall sliding: first X-only movement, then Y-only. If completely blocked, the camera stays put.

RAY TURN degrees!

Rotate the camera heading.

Parameter	Description
degrees!	Rotation amount (positive = clockwise/right, negative = left)

The heading is automatically normalised to 0-360 degrees.

RAY CELL x, y, value

Write a value to a map cell at runtime.

Parameter	Description
x	Cell X coordinate (0 to map width - 1)
y	Cell Y coordinate (0 to map height - 1)
value	Wall type (0 = empty, 1-31 = wall)

Use this for doors, destructible walls, switches, or any dynamic map modification.

RAY CAST angle!

Cast a single ray from the camera position at an absolute angle.

Parameter	Description
angle!	Absolute angle in degrees

Results are stored internally and retrieved with RAY(CASTDIST), RAY(CASTWALL), RAY(CASTSIDE), RAY(CASTX), RAY(CASTY).

Useful for "use" buttons, shooting mechanics, proximity checks, or finding what the player is looking at.

RAY SPRITE id, spritenum, x!, y!

Place or update a billboard sprite in the world.

Parameter	Description
id	Raycaster sprite slot (0-31)
spritenum	SPRITE buffer number (1-64)
x!, y!	World-space position (floating-point)

The sprite's full-colour 4bpp image is read from the sprite buffer and rendered as a billboard (always facing the camera). Pixels matching the current SPRITE TRANSPARENT colour are not drawn. Sprites are automatically depth-sorted and clipped against the wall z-buffer.

Note: The sprite buffer must be loaded before calling RAY SPRITE. Use SPRITE LOADARRAY to create sprites from BASIC arrays, or SPRITE LOAD to load .spr files.

RAY SPRITE REMOVE id

Remove sprite id from the raycaster (stop rendering it).

RAY SPRITE CLEAR

Remove all sprites.

RAY MINIMAP x, y, size

Draw a top-down minimap overlay onto the framebuffer.

Parameter	Description
x	Screen X position of the minimap
y	Screen Y position of the minimap
size	Size in pixels (longest map axis fits within this)

The minimap scales the entire map to fit, preserving aspect ratio. Each wall cell is drawn in that wall definition's foreground colour. Door cells show their definition's foreground colour when closed, yellow when partially open, and black when fully open. Empty space is black, the player is a white dot with a direction indicator, and sprites are yellow dots. A dark green border is drawn around the edge.

Call after RAY RENDER and before FRAMEBUFFER COPY.

RAY DOOR x, y, offset!

Set a sliding door at a map cell with a given open offset.

Parameter	Description
x	Cell X coordinate (0 to map width - 1)
y	Cell Y coordinate (0 to map height - 1)
offset!	Door offset: 0.0 = fully closed, 1.0 = fully open

The map cell at (x, y) must contain a wall type whose definition has the door flag set (see RAY DEFINE). When offset is between 0 and 1, the door is partially open - rays pass through the open portion and hit the remaining solid portion. When offset reaches 1.0, the cell becomes fully passable for both rays and movement.

Up to 8 doors may be active simultaneously. To animate a door, call RAY DOOR each frame with an incrementally increasing or decreasing offset.

RAY DOOR CLOSE x, y

Remove the door slot at (x, y). The cell reverts to a normal solid wall. Call this after closing animation completes (offset reaches 0.0).

RAY DOOR CLEAR

Remove all active door slots.

RAY DEFINE type, fg, bg, pattern [, door]

Set the visual properties for a wall type.

Parameter	Description
type	Wall type to define (1-31)
fg	Foreground colour (RGB121 palette index 0-15)
bg	Background colour (RGB121 palette index 0-15)
pattern	Fill pattern index (0-31)
door	Optional. 1 = door, 0 = normal wall (default 0)

Every wall type from 1 to 31 has a definition that controls its rendered appearance. The definition is used for both X-side and Y-side hits; Y-side hits are automatically dimmed by the engine for depth cueing.

Definitions persist until RAY CLOSE. Call RAY DEFINE after RAY MAP but before RAY RENDER.

Defaults (set by RAY MAP):

Types	fg	bg	pattern	door
1-15	GREEN (6)	MIDGREEN (4)	type - 1	0
16-31	YELLOW (14)	BROWN (12)	type - 1	1

Example - red brick walls for type 1:

```
RAY DEFINE 1, 8, 10, 3       ' fg=RED, bg=RUST, pattern 3
```

Example - custom blue door:

```
RAY DEFINE 7, 1, 3, 6, 1     ' fg=BLUE, bg=COBALT, pattern 6, door=1
RAY CELL door_x, door_y, 7
```

RAY RENDER

Render the complete 3D scene into the current WriteBuf framebuffer. This draws:

1. Textured floor and ceiling (horizontal scanlines)
2. Textured walls (vertical columns via DDA raycasting)
3. Billboard sprites (depth-sorted, z-buffered)

A framebuffer must be active (FRAMEBUFFER CREATE / FRAMEBUFFER WRITE F).

RAY CLOSE

Free all raycaster state (map, column arrays, sprites). Called automatically on program end.

Functions

All functions return values from the current raycaster state.

Function	Returns	Type
RAY(MAPW)	Map width	Integer
RAY(MAPH)	Map height	Integer
RAY(CAMX)	Camera X position	Float
RAY(CAMY)	Camera Y position	Float
RAY(CAMA)	Camera angle (degrees)	Float
RAY(DIST col)	Perpendicular wall distance at column col	Float
RAY(WALL col)	Wall type hit at column col	Integer
RAY(CELL x, y)	Map cell value at (x, y)	Integer
RAY(DOOR x, y)	Door offset, or -1.0 if not active	Float
RAY(CASTDIST)	Distance from last RAY CAST	Float
RAY(CASTWALL)	Wall type from last RAY CAST	Integer
RAY(CASTSIDE)	Side hit (0=X-side, 1=Y-side)	Integer
RAY(CASTX)	Map X cell from last RAY CAST	Integer
RAY(CASTY)	Map Y cell from last RAY CAST	Integer
RAY(SPRITES)	Count of active sprites	Integer
RAY(SPRITEX id)	World X of sprite id	Float
RAY(SPRITEY id)	World Y of sprite id	Float
RAY(DEFINE t, p)	Wall def field: 0=fg,1=bg,2=pat,3=door	Integer

RGB121 Colour Palette

The 4-bit RGB121 palette provides 16 colours. Each index encodes 1 bit red, 2 bits green, 1 bit blue.

Index	Name	RGB888	Appearance
0	BLACK	000000	Black
1	BLUE	0000FF	Blue
2	MYRTLE	004000	Dark green
3	COBALT	0040FF	Dark cyan-blue
4	MIDGREEN	008000	Medium green
5	CERULEAN	0080FF	Sky blue
6	GREEN	00FF00	Bright green
7	CYAN	00FFFF	Cyan
8	RED	FF0000	Red
9	MAGENTA	FF00FF	Magenta
10	RUST	FF4000	Dark orange
11	FUCHSIA	FF40FF	Pink
12	BROWN	FF8000	Brown/orange
13	LILAC	FF80FF	Light pink
14	YELLOW	FFFF00	Yellow
15	WHITE	FFFFFF	White

Wall Definitions

Every wall type from 1 to 31 has a fully customisable wall definition comprising a foreground colour, background colour, fill pattern, and a door flag. Use RAY DEFINE to set these; the defaults are:

Types	Foreground	Background	Pattern	Door
1-15	GREEN (6)	MIDGREEN (4)	type - 1	0 (no)
16-31	YELLOW (14)	BROWN (12)	type - 1	1 (yes)

Y-side depth cueing: When a wall column is a Y-side hit, the foreground and background colours are automatically dimmed by decrementing the green channel of the RGB121 value (e.g., GREEN 6 -> MIDGREEN 4, YELLOW 14 -> BROWN 12). This gives walls a bright/dark shade difference that conveys depth without requiring a separate colour definition per side.

Fill Patterns

Wall and floor/ceiling textures use the Turtle graphics fill patterns (indices 0-31). Each pattern is an 8x8 grid of 1-bit pixels. During rendering, pattern bit 1 selects the foreground colour and bit 0 selects the background colour.

Each wall definition specifies a pattern index (0-31) via RAY DEFINE. By default, pattern = wall_type - 1, giving each wall type a unique texture. Patterns are tiled twice per map cell for increased texture density.

Sprite System

Raycaster sprites use the standard PicoMite SPRITE system for image data. Sprites are loaded into sprite buffers (1-64) using SPRITE LOAD or SPRITE LOADARRAY, then placed into the raycaster world using RAY SPRITE.

Loading Sprites from Arrays

```
' Define a 16-colour 8x8 sprite using RGB888 colour values
DIM INTEGER pixels%(63)
' ... fill pixels% with RGB888 colours ...
SPRITE LOADARRAY buffer_num, width, height, pixels%()
```

Each pixel is an RGB888 value (e.g., &hFF0000 for red). The system automatically converts to the 4-bit RGB121 palette. Pixels are packed two per byte (even pixel in low nibble, odd pixel in high nibble).

Transparency

```
SPRITE TRANSPARENT colour_index
```

Default is 0 (BLACK). Any sprite pixel matching this index is not drawn, allowing the wall/floor behind to show through.

Sprite Rendering

During RAY RENDER, sprites are:

1. Sorted by distance from camera (furthest first - painter's algorithm)
2. Projected onto the screen as billboards (always face the camera)
3. Scaled based on distance, preserving the sprite's aspect ratio
4. Clipped per-column against the wall z-buffer (walls occlude sprites)
5. Drawn pixel-by-pixel, skipping transparent pixels

Up to 32 raycaster sprites can be active simultaneously, referencing up to 64 sprite buffers.

Implementing Doors

Doors use the RAY DOOR command for smooth sliding animation. Any wall type can act as a door if its definition has the door flag set (RAY DEFINE type, fg, bg, pattern, 1). By default, types 16-31 have the door flag.

Basic Door Setup

```
' Place a door wall (type 31 has door flag set by default)
RAY CELL door_x, door_y, 31

' Or define a custom door type (e.g. red/rust with pattern 7)
RAY DEFINE 5, 8, 10, 7, 1
RAY CELL door_x, door_y, 5
```

Animated Opening

```
' Start opening: create a door slot and animate over multiple frames
door_offset! = 0.0
DO WHILE door_offset! < 1.0
  door_offset! = door_offset! + 0.1
  IF door_offset! > 1.0 THEN door_offset! = 1.0
  RAY DOOR door_x, door_y, door_offset!
  RAY RENDER
  FRAMEBUFFER COPY F, N
  PAUSE 50
LOOP
' Door is now fully open - player can walk through
```

Animated Closing

```
' Close: animate offset back to 0, then release the slot
DO WHILE door_offset! > 0.0
  door_offset! = door_offset! - 0.1
  IF door_offset! < 0.0 THEN door_offset! = 0.0
  RAY DOOR door_x, door_y, door_offset!
  RAY RENDER
  FRAMEBUFFER COPY F, N
  PAUSE 50
LOOP
RAY DOOR CLOSE door_x, door_y
' Door is fully closed and slot is released
```

How It Works

When RAY DOOR sets an offset between 0 and 1, the door slides open from one side. Rays that hit the open portion pass through to the corridor behind; rays that hit the remaining solid portion render the door texture. The door blocks movement until offset reaches 1.0, at which point the cell becomes fully passable.

The minimap shows door state: the definition's foreground colour when closed, yellow when partially open, empty when fully open.

Coordinate System

- Map coordinates are 0-based. Cell (0,0) is the top-left corner.
- X axis increases to the right (east).
- Y axis increases downward (south).

- Angle 0 degrees points in the +X direction (east). Angles increase clockwise: 90 = south, 180 = west, 270 = north.
- Camera and sprite positions use floating-point coordinates. A position of (2.5, 3.5) is the centre of cell (2, 3).

Typical Game Loop

```
MODE 2
CLS
' ... define map array ...
FRAMEBUFFER CREATE
FRAMEBUFFER WRITE F

RAY MAP w, h, map%()
RAY CAMERA start_x, start_y, start_angle, 66
RAY COLOUR floor_fg, ceil_fg, floor_bg, ceil_bg, floor_pat, ceil_pat

' ... load sprites, place them ...

DO
  k$ = INKEY$
  IF k$ = CHR$(27) THEN EXIT DO

  ' Movement
  IF k$ = "w" THEN RAY MOVE 0.15
  IF k$ = "s" THEN RAY MOVE -0.15
  IF k$ = "a" THEN RAY TURN -5
  IF k$ = "d" THEN RAY TURN 5

  ' Interaction
  IF k$ = " " THEN
    RAY CAST RAY(CAMA)
    IF RAY(CASTDIST) < 2.0 THEN
      ' ... handle what was hit ...
    ENDIF
  ENDIF

  ' Render
  RAY RENDER
  RAY MINIMAP 2, 2, 48
  FRAMEBUFFER COPY F, N
LOOP

RAY CLOSE
FRAMEBUFFER CLOSE
```

Limitations

- RP2350 only - not available on RP2040 builds.
- Maximum map size: 256 x 256 cells.
- Maximum raycaster sprites: 32 active at once.
- Maximum active doors: 8 simultaneously.
- Maximum sprite buffers: 64 (shared with the standard SPRITE system).
- Wall types: 1-31 only (capped at the number of Turtle fill patterns).
- Colour depth: 4-bit RGB121 (16 colours). All 16 colours are available for walls (via RAY DEFINE), floor, ceiling, and sprites.
- Resolution: Uses the current HRes and VRes - works at any resolution. Mode 2 (320x240) is recommended; higher resolutions work but render more slowly.
- Single height level: No floor/ceiling height variation (classic Wolfenstein-style).

Appendix A: Technical Implementation

Architecture

The raycaster is implemented entirely in C (Raycaster.c, ~1600 lines) with a header (Raycaster.h). It integrates into the MMBasic command/function dispatch system via cmd_ray() and fun_ray(), registered in AllCommands.h. State cleanup is hooked into CloseAllFiles() in FileIO.c.

All state is held in a single heap-allocated RayState structure, accessed through the static pointer rstate. Memory is managed using PicoMite's GetMemory()/FreeMemory() allocator.

DDA Algorithm

The wall-casting engine uses the standard Digital Differential Analyzer (DDA) algorithm:

1. For each screen column, a ray is cast from the camera position through the corresponding point on the camera plane.
2. The ray steps through the grid one cell boundary at a time, alternating between X-side and Y-side crossings, always choosing the nearer crossing.
3. When the ray enters a cell with a non-zero wall type, the DDA loop terminates.
4. The perpendicular distance (not Euclidean) is computed to avoid fisheye distortion: perp_dist = side_dist - delta_dist for the last step.
5. The wall strip height is screen_height / perp_dist.

The perpendicular distance for each column is stored in col_dist[] and reused as the z-buffer for sprite clipping.

Wall Rendering

Walls are drawn as textured vertical strips using ray_vline_textured(). Each wall type selects a Turtle fill pattern (8x8, 1-bit). The texture coordinates are:

Horizontal (tex_x): Derived from the fractional wall-hit position, multiplied by 16 and masked to 0-7. This tiles the 8-texel pattern twice across each wall face.

Vertical (tex_y): Mapped from the screen Y range with a 16x multiplier and & 7 wrapping, tiling the pattern twice vertically per wall height.

Pattern bits select between a foreground/background colour pair. Each wall type has a configurable foreground/background colour pair and fill pattern, set via RAY DEFINE. Y-side hits are automatically dimmed by decrementing the RGB121 green channel, providing a depth cue without requiring separate per-side colours.

Door Rendering

Sliding doors are handled within the DDA loop. When a ray enters a cell whose wall definition has the door flag set, the engine checks whether an active door slot exists for that cell:

1. If the door offset is 0.0 (or no slot exists), the cell is treated as a normal solid wall.
2. If the door offset is >= 1.0, the cell is fully open - the ray passes through and the DDA continues to the next cell.
3. For intermediate offsets (0.0-1.0), the engine computes the fractional hit position on the wall face. If the fractional position is less than the door offset, the ray passes through the open portion. Otherwise, the ray hits the remaining solid

door.

This creates a sliding-door effect: the opening grows from one side of the wall face as the offset increases. Collision detection also respects door state: cells with a door offset >= 1.0 are passable; partially-open doors still block movement.

Up to 8 doors can be active simultaneously, stored in fixed-size slots within RayState.

Floor and Ceiling Rendering

The floor/ceiling renderer uses a horizontal scanline approach, which is more efficient than per-pixel ray casting:

1. For each row below the horizon, the row distance is calculated: row_dist = half_screen_height / (row - horizon).
2. The world-space floor position at the leftmost and rightmost screen edges is computed using the camera's left and right ray directions.
3. A linear interpolation step is computed per column, and the inner loop advances by addition only (no per-pixel division).
4. Texture coordinates are computed by multiplying the world position by 16 and masking to 0-7, tiling the pattern twice per map cell.
5. The ceiling row is mirrored: ceil_y = screen_height - 1 - floor_y.
6. Pixels are written two at a time (even/odd nibble packing) to minimise memory operations.

Sprite Rendering

Billboard sprites are rendered after walls, using the wall z-buffer for per-column occlusion:

1. Active sprites are collected and sorted by squared distance from the camera (furthest first - painter's algorithm).
2. Each sprite's world position is transformed into camera space using the inverse of the 2x2 camera matrix (direction x plane).
3. The sprite is projected to a screen rectangle based on its distance, with width scaled by the sprite image's aspect ratio.
4. For each visible screen column (not occluded by a closer wall), the sprite's 4bpp pixel data is sampled from the SPRITE buffer.
5. Pixels matching sprite_transparent are skipped. Other pixels are written directly into the framebuffer.

The 4bpp pixel format matches the framebuffer layout: even pixels in the low nibble, odd pixels in the high nibble of each byte.

Collision Detection

RAY MOVE implements collision detection with wall sliding:

1. A 0.25-unit radius bounding box is checked at the target position.
2. All four corners of the box are tested against the map grid.
3. If any corner overlaps a wall cell, the full move is blocked.
4. The engine then tries X-only movement (slide along Y walls).
5. If that's also blocked, it tries Y-only movement (slide along X walls).
6. If completely blocked, the camera doesn't move.

RAY CAST

RAY CAST runs the same DDA algorithm as the main renderer but for a single ray at an arbitrary angle. Results are stored in the cast_* fields of the raycaster state and queried via RAY(CASTDIST), RAY(CASTWALL), RAY(CASTSIDE), RAY(CASTX), RAY(CASTY).

Minimap

The minimap iterates over all map cells, scaling to fit the longest axis within the specified pixel size while preserving the map's aspect ratio. Active sprites are drawn as coloured dots. The player is shown as a white dot with a 2-pixel direction indicator computed from the camera angle.

Memory Usage

Item	Size
RayState structure	~2.6 KB (32 wall defs, 32 sprites, 8 doors)
Map storage	w x h bytes (max 64 KB for 256x256)
Column distance array	HRes x 4 bytes (1280 for 320 cols)
Column wall-type array	HRes bytes (320)

Total overhead for a typical 57x51 map at 320x240: approximately 6.5 KB plus the framebuffer.

Compilation

The raycaster is compiled with -Os (optimise for size) across all RP2350 build variants. It is conditionally included via #ifdef rp2350 in AllCommands.h and CMakeLists.txt.

Appendix B: Demo Program

The following program demonstrates all raycaster features in a continuous automated walkthrough. It defines a 57x51 map, creates five colourful sprites, builds a wall with an animated sliding door, and runs a pre-programmed round-trip sequence that loops indefinitely.

Demo Code

```
' Raycaster Demo for PicoMite MMBasic (RP2350)
' Continuous auto-play loop with animated sliding door
' Press ESC at any time to quit
OPTION EXPLICIT
MODE 2
CLS

CONST MAP_W = 57
CONST MAP_H = 51
CONST FOV = 66
CONST START_X = 25.5
CONST START_Y = 23.5
CONST START_A = 0

DIM INTEGER x%, y%, i%, cx%, cy%, door_x%, door_y%, door_wall%
DIM k$
DIM FLOAT moveSpeed, rotSpeed
moveSpeed = 0.2
rotSpeed = 5.625   ' exactly 90 degrees per 16 steps

DIM FLOAT door_offset!, door_target!, door_step!
DIM INTEGER door_animating%
door_step! = 0.1

DIM INTEGER world%(MAP_W * MAP_H - 1)
RESTORE MapData1
FOR y% = 0 TO MAP_H - 1
  READ k$: k$ = k$ + "1"
  FOR x% = 0 TO MAP_W - 1
    world%(y% * MAP_W + x%) = VAL(MID$(k$, x% + 1, 1))
  NEXT x%
NEXT y%

' Vary wall types for texture variation
FOR y% = 0 TO MAP_H - 1
  FOR x% = 0 TO MAP_W - 1
    IF world%(y% * MAP_W + x%) > 0 THEN
      world%(y% * MAP_W + x%) = ((x% + y%) MOD 5) + 1
    ENDIF
  NEXT x%
NEXT y%

FRAMEBUFFER CREATE
FRAMEBUFFER WRITE F
RAY MAP MAP_W, MAP_H, world%()
RAY COLOUR 12, 3, 8, 1, 1, 3

' ... (sprite creation and placement code) ...

DIM seq$
seq$ = "P5W22P3O1P10W15P5D16W5P5W5D16W30D16"
seq$ = seq$ + "W5P5W5P5D16W15P3C1P10A16W5P5D32"
```

```
seq$ = seq$ + "W5D16W22D16D16P5"

DO  ' Main loop - runs continuously until ESC
  RAY CAMERA START_X, START_Y, START_A, FOV
  ' ... rebuild door wall, reset animation state ...
  ' ... execute sequence with per-frame rendering ...
  PAUSE 500
LOOP
```

How the Demo Works

Initialisation

The demo starts by setting Mode 2 (320x240 @ 4bpp RGB121), then reads a 57x51 map from DATA statements. Each DATA line is a string of digits where 1 = wall and 0 = empty. After reading, wall cells are varied to types 1-5 using a (x + y) MOD 5 + 1 formula, ensuring different fill patterns appear across the map.

A framebuffer is created and the raycaster is initialised with RAY MAP, RAY CAMERA, and RAY COLOUR (brown floor pattern 1, cobalt ceiling pattern 3).

The rotation speed is set to exactly 5.625 degrees per step, so that D16 (16 right-turn steps) makes exactly 90 degrees and four such turns return to the original heading - essential for the seamless loop.

Door Construction

Six RAY CELL commands build a north-south wall segment at x=30 (rows 19-24). Five cells use wall type 3 (green by default), while the cell at y=23 uses type 31 (brown/yellow with door flag by default). From the starting position, this wall is clearly visible ahead with the door panel standing out.

Sprite Creation

Five 8x8 sprites are created procedurally using SPRITE LOADARRAY:

Buffer	Design	Method
1	Red cross	2-pixel-wide cross at cols 3-4 and rows 3-4
2	Yellow diamond	Manhattan distance from centre <= 3
3	Green/cyan stripes	Alternating columns
4	Magenta/blue check	(x + y) MOD 2 test
5	White ring	Border pixels and corner diagonals

Door Animation System

The demo uses per-frame door animation driven by BASIC variables:

door_offset! - current door position (0.0-1.0)
door_target! - where the door is heading (0.0 or 1.0)
door_animating% - whether animation is active
door_step! - offset change per frame (0.1 = 10 frames to fully open/close)

The O command sets door_target! = 1.0 and starts animating. Each frame, if door_animating% is set, the offset moves toward the target by door_step!. When the target is reached, animation stops. For closing (C command), the same logic runs in reverse; when offset reaches 0.0, RAY DOOR CLOSE releases the slot.

Sequence Engine

The auto-play system uses a compact string encoding: each command is a single letter followed by a repeat count.

Letter	Action
W	Walk forward (RAY MOVE moveSpeed)
S	Walk backward (RAY MOVE -moveSpeed)
A	Turn left (RAY TURN -rotSpeed)
D	Turn right (RAY TURN rotSpeed)
O	Start door open animation
C	Start door close animation
P	Pause (render without moving)

The parser reads one letter, then digits for the repeat count (e.g., W22 = walk forward 22 steps, D16 = turn right 16 steps = 90 degrees). Each step renders a frame, draws the minimap, adds a crosshair, and copies the framebuffer to screen with a 50ms delay.

Continuous Loop

The entire demo is wrapped in an outer DO...LOOP. At the top of each iteration, the camera is reset to the start position (25.5, 23.5) facing east, the door wall is rebuilt via RAY CELL commands, door animation state is cleared, and the sequence replays from the beginning. This creates a seamless continuous demonstration. Pressing ESC exits cleanly via GOTO Done.