(back to project list)

Missile Command (1980) Disassembly

Missile Command is an arcade game in which the player defends cities from nuclear attack by firing counter-missiles. The game's theme is a reflection of Cold War fears in the 1980s.

Competitions for the highest score come in two forms. In "marathon mode", extra cities are awarded every 10,000 points. In "tournament mode", extra cities are disabled entirely. A single marathon game can last for 2 or 3 days.

- **Disassembly listing** of the Missile Command rev 3 ROMs.
- **Project set** - project files and binaries (requires SourceGen v1.8 or later).
- **Attack wave details**, including colors, speeds, and missile counts.
- **Video memory** layout, including a description of the peculiar 2bpp/3bpp frame buffer and linearized MADSEL memory access.
- Explanation of **revision 2** ROM set differences from revision 3, with a full explanation of a couple of significant bugs.
- List of **messages** displayed by the program, handy if you're trying to figure out what the code is doing.
- Instructions for **creating the binary** used for the disassembly listing.

Related resources:

- **Original sources**. The original source code for the revision 1 ROM is now available.
- **MAME**. The best way to play classic arcade games if you can't afford the real thing.
- **Atari service manuals** at arcarc.xmission.com. Includes manuals for various versions of Missile Command.
- **Video of a talk** by Tony Temple, tournament world record holder, about the game and its history (from Free Play Florida 2018). You can see a tape of his 2010 tournament-mode world record game here.
- Fun article about the history of **trackballs** in arcade games.

Game Play

A few definitions, for those of you who didn't live through the 1980s. Incoming missiles are referred to as "ICBMs" (Inter-Continental Ballistic Missile), missiles fired by the player as "ABMs" (Anti-Ballistic Missile), and ICBMs that split as "MIRVs" (Multiple Independent Reentry Vehicle). Bombers and killer satellites are collectively "fliers". These may not be entirely accurate, but it's less ambiguous than just calling everything "missile".

Using MAME

The easiest way to play is via emulation on a computer. Once you have the Missile Command ROM file, save it in the MAME "roms" folder". When you start the MAME UI front-end, it will find the game automatically. Select it from the list and hit Enter. When the system summary displays, hit Enter again. (If this doesn't work for you, please visit one of the many MAME help sites.)

While the game is running, you can get a list of keys by hitting Tab, selecting "Input (this Machine)" with the arrow keys, and hitting Enter. The most important are:

- `5` - deposit a coin
- `1` - start game (1 player)
- `LeftArrow` / `UpArrow` / `RightArrow` / `DownArrow` - move crosshairs
- `Ctrl` / `Alt` / `Space` - fire from left / center / right launcher

- `P` - pause/unpause
- `ESC` - exit

Hit `Tab` to open a menu with a "DIP Switches" sub-menu that will let you set the machine for free play so you can stop inserting virtual quarters.

If you don't want to move the crosshairs with the arrow keys, you'll want to configure MAME to use a mouse or game controller. From the main MAME menu, select Configure Options, then Device Mapping, then adjust the Trackball Device Assignment. If you select "mouse", you can use the mouse to move the pointer and the left, middle, and right buttons to fire. Note that selecting the mouse as the controller will cause MAME to grab the mouse cursor, which is fine while playing full-screen but inconvenient while trying to use the debugger.

I used MAME v0.236 for my testing.

Playing

ICBMs can drop from the top of the screen or be launched by fliers, which move horizontally about halfway up the screen. Incoming missiles move slowly on the first wave, but speed up as the game progresses, and become more numerous. (See the Wave Guide page for details.) After a few waves, smart bombs start to appear. These move at the same speed as missiles, but are able to change direction to dodge around explosions.

Use the trackball to position the crosshairs, and the three fire buttons to launch ABMs from the left, center, or right missile silo. There are ten ABMs per silo. ABMs launched from the side silos move more slowly (3 units per frame) than those launched from the center silo (7 units per frame), making the center silo more effective for last-minute intercepts.

ICBMs that reach the area at the bottom of the screen will destroy a city or missile silo. Silos are fully restored at the start of each wave.

The wave ends after a certain number of missiles and smart bombs have been destroyed.

Points are awarded for destroying attackers as follows:

Hostile Entity	Points
Missile	25
Satellite	100
Bomber	100
Smart Bomb	125

In addition, at the end of each wave the game awards 5 points for un-fired ABMs, and 100 points for each surviving city. All points are multiplied by the current scoring multiplier, which starts at 1x and increases every other level until it reaches 6x at wave 11.

Bonus cities are awarded every N points, where N is set with physical DIP switches in the cabinet (default 10,000). When a city is destroyed, it's replaced with a bonus city if any are available. You can have a large number of bonus cities pending.

Details

The game has 16 slots for missiles: 8 for ABMs, and 8 for ICBMs/bombs. There's one slot for a flier. If you try to fire more than 8 ABMs at once the game will refuse. Explosions don't hold on to a missile slot, but the number of active explosions is factored into decisions like whether or not to launch more ICBMs.

The flier has a resurrection cooldown and a firing cooldown that become shorter on later waves. Fliers can fire multiple missiles at once. The flier's type, altitude, and direction of travel are random.

Smart bombs share most of the data structures with ICBMs, which means they move at the same speed and are counted against the limit of 8 simultaneous attacks. The game also limits itself to having only two smart bombs on the screen at any time, and in some calculations smart bombs count as two missiles.

Smart bombs use the same movement routine that missiles do, unless they detect an explosion nearby. When that happens, movement is controlled by a table-driven routine that tries to move the smart bomb toward the target without driving it closer to the explosion. The anti-collision routine works by reading screen pixels and looking for the flashing color (#4/#5), which means it can react to the flashing 'X' that is placed when firing an ABM, or to a passing missile head.

While bombs can move evasively, they do not change targets.

Attack Limitations

The code that picks targets ($5791) is designed so that the player will never lose more than 3 cities in a single wave. For example, if you have lost two cities, the remaining missiles will not target more than one city at a time. (If you destroy all missiles aimed at city A, it might throw some new ones at city B, but it won't have missiles in flight to both A and B.)

If no ABMs are left, and 3+ cities have been destroyed, the wave will end immediately with no further attacks even if there are pending or in-flight ICBMs and bombs. There's no reason to continue, because the incoming missiles or bombs can't have any effect, and with no ABMs there's no way to score points. (See $59fa)

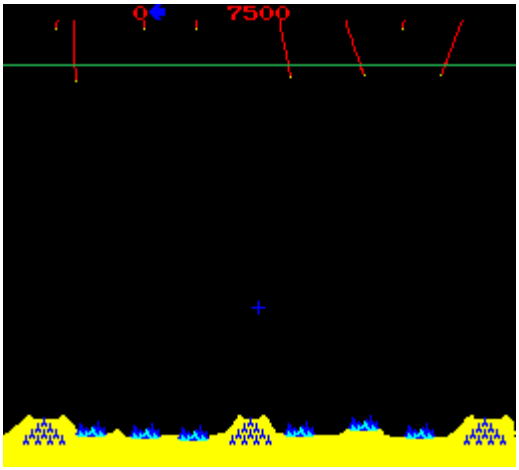
An ICBM can split (MIRV) on any wave if certain conditions are met. The code walks through the 8 ICBM slots, updating a maximum altitude value as it goes. If the maximum altitude seen so far is between 128 and 159, meaning we have at least one ICBM in the upper-middle of the screen and nothing higher, then the ICBM currently being examined becomes a MIRV candidate. The candidate's altitude is irrelevant, except that it must be below 160. So the conditions required for an ICBM to be eligible to split are (see $5379/$56d1):

- The current missile, or a previously-examined missile, is at an altitude between 128 and 159.
- No previously-examined missile is above 159.
- There must be available slots in the ICBM table, and unspent ICBMs for the wave.

A single missile can spawn up to 3 additional missiles. Each missile will have a different target.

A more direct use of the maximum ICBM height is used to pace the attack. The game doesn't launch any new attacks while the highest ICBM is above `(202 - 2 * wave_number)`, with a minimum value of 180. At lower waves this means that missiles will trickle in steadily, while at higher waves the game will wait until everything is below 180 before sending any more. The calculation is mildly broken in rev 2 ROMs, causing a change in behavior at wave 102.

This effect can be observed in wave 1. The initial launch sends four ICBMs at the player, which is the maximum that can be fired in one frame. In the next frame the altitude of the highest ICBM is well above 200, so no new missiles can be launched. Once the highest missile is below that point, another four ICBMs are launched. To provide a reference point, this image has a green line drawn at row 200:



Bonus Cities

Bonus cities are awarded every N points, where N is determined by a DIP switch setting inside the cabinet. It varies from 8,000 to 20,000, and can be disabled entirely. (Table at $6082.) The number of initial cities can also be set, from 4 to 7. (Table at $5b08.) Marathon rules specify 6 initial cities with a bonus every 10,000 points.

Multiple bonus cities may be awarded on a single wave. For example, on wave 19, if the player destroys all 22 ICBMs and 7 smart bombs, they would receive `((22 * 25) + (7 * 125)) * 6 = 8550` points. Bonus points for surviving cities would be `(6 * 100) * 6 = 3600` if all cities survived. That adds up to 12150, and could be higher if the player destroyed fliers and didn't fire every ABM. So receiving multiple bonus cities won't happen every wave, but it can happen.

The game tracks the total number of cities held by the player, including those on screen, in a single 8-bit value. The number increases as bonus cities are awarded, and decreases as destroyed cities are replaced. If the count reaches 256, it rolls over to zero.

The position of replacement cities, i.e. which crater gets filled, is chosen at random.

Significant Bugs

The revision 2 ROM has a couple of bugs that dramatically impact scoring, and arguably make "marathon mode" possible. The most important is the "810 bug", where the game awards 176 bonus cities when the score reaches 800,000 points plus the bonus city score.

In a marathon game, the player needs to carefully manage the total number of cities they have so that it's below 80 when they reach 810K points. This can allow them to have up to 255 cities. Since the game will reliably destroy 3 cities per wave, and an unattended wave lasts about 15 seconds, the player could take a break for as long as an hour if the city count were maximized. The game doesn't tell you how many bonus cities you have banked, however, so keeping count is a significant part of the strategy.

The most entertaining bug happens at wave 255 and 256, when the bonus multiplier jumps from 6x to 256x. Wave 256 is completely devoid of enemy attack; the wave immediately ends, with bonus points awarded for all 30 unfired ABMs, and up to 6 surviving cities. The huge multiplier provides a nice boost to the score, even on the empty wave 256: `(30 * 5 + 6 * 100) * 256 = 192,000`.

See the rev 2 page for a detailed explanation of the bugs, which were fixed in revision 3.

Digging Deeper

Let's start with a map of main memory (cf. the Missile Command hardware driver source code in MAME).

Addr Range	Description
0000-01ff	General-purpose RAM (incl. stack)
0200-05ff	Video RAM, 3rd color bit for last 32 lines
0600-063f	General-purpose RAM
0640-3fff	Video RAM, 2bpp
4000-400f	POKEY I/O
4800	IN0 (read), two modes
4800	OUT0 (write)
4808	R8 DIP switches (read)
4900	IN1 (read)
4a00	R10 DIP switches (read)
4b00-4b07	Color palette (write)
4c00	Watchdog reset (write)
4d00	IRQ ack (write)
5000-7fff	Game ROM

Most of the stack page is used for game state. Of the 16KB of RAM, all but 576 bytes is used for video memory.

The display refreshes about 60x per second. IRQs fire four times per frame (~240Hz), and the main game thread is woken up every four IRQs, so the main game logic is designed around a 60Hz update rate.

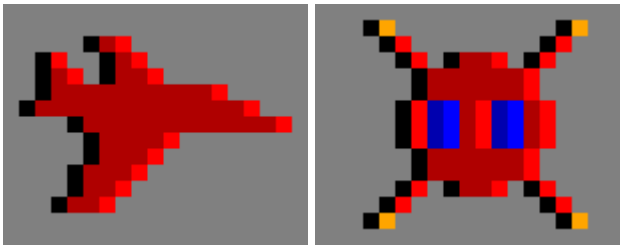
The IRQ handler reads the controls, manages the coin slots and sound effect generation, and increments some of the color palette entries to produce the flashing effect.

Video memory is interesting. Most of the screen uses two bits per pixel, but the bottom 32 lines use 3 bits per pixel. The details of how this works are described in the video memory page.

The game is available in a cocktail table cabinet, with players on opposite sides of the table in two-player games. The 180-degree rotation of the screen is done partly in hardware (for the vertical flip) and partly in software (for the horizontal flip). The hardware provides separate I/O bits for the second set of fire buttons, but delivers the second trackball through the same I/O locations as the first.

Animation

The game does very little drawing on each frame. Missiles require drawing only two pixels each: one to draw the missile head's new position, one to draw over the previous position with the trail color. Smart bombs are fully erased and redrawn, but they're limited to two at a time. Fliers are never fully drawn or erased, as they use a renderer that draws the leading edge and erases the trailing edge, and rely on the explosion to erase the flier.



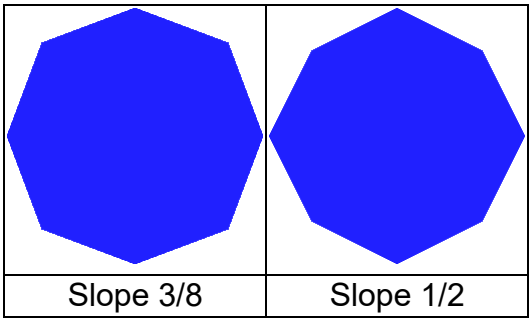
(Pixels that are plotted are shown in bright colors, while the pixels that are left over from a previous frame are shown in dim colors. Black pixels represent the background color. See $61ed.)

If the game isn't able to complete all of its updates and rendering in the first 1/4 of a frame, it will defer redrawing the score. This is a reasonable optimization because, if there's a lot going on, the player is probably not focused on the score. (See $50ff.)

Explosions are drawn as a series of octagons, not circles, expanding over several frames and then shrinking over several. There are slots for 20 explosions. To slow the rate at which they expand and contract, and also to reduce the per-frame load, they're divided into 5 groups of 4, and only one group is updated on a given frame.

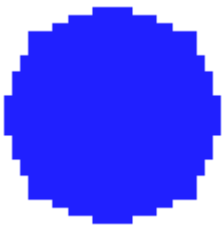
Side note: collision testing is performed when an explosion is drawn, every 5 frames, so it's possible for a fast-moving ICBM to pass through a small explosion. Only ICBMs are tested for collisions, so ABMs can pass through explosions unharmed. Collision testing isn't performed below line 33, which is why you can have multiple ICBMs slamming into a city that is already engulfed in a fireball.

The angles on the octagon are uneven. The lines are rendered in the first octant with a slope of 3/8 instead of 1/2, which makes it look less like a tilted USA stop sign:



Slope 3/8	Slope 1/2

Being closer to square also makes "THE END" fit more easily in the game-over screen. The only negative impact of the shape is that it causes diagonal spikes on the explosion when it reaches its maximum radius (13):



Much of the sense of motion in the game is conveyed through color cycling. One entry (technically two entries in the 3bpp area) is rotated through all 8 possible colors at 30Hz. This is done in the IRQ handler so that the flashing is consistent even when things get busy. During the initial title sequence, when the words "MISSILE COMMAND" are getting blown up, the game is erasing and redrawing all

20 explosions, which takes several game frames. The fact that it's a little sluggish isn't noticeable because all the colors are cycling.

The missile trails aren't plotted with the standard Bresenham line-drawing algorithm. Instead, the game computes the distance from the missile's entry point to its target, then divides the X and Y components by the distance to get a fractional increment (held as a fixed-point signed 8.8 value). Every time the missile moves, the increments are added, and the new point is plotted. The resulting lines can look a little ragged.

The missile won't necessarily strike the exact target, due to various inaccuracies in the math: the distance is calculated with an approximation; the distance is capped at 255; and minor errors in the fixed-point increment add up over time. The game handles this by advancing the missile until it moves *past* the target in the X or Y axis. This effect isn't noticeable in-game, suggesting that "close enough" is fine for horseshoes, hand grenades, and nuclear weapons.

The scrolling text shown at the bottom of the screen in attract mode uses colors 0 and 1, which are the main background and the city background. Because the colors only differ in the 3rd bit, the scrolling can be done with a read/modify/write operation in the part of video RAM that holds the third bit.

Sound

Sound effects are generated by the Pot Keyboard Integrated Circuit ("POKEY") chip, which provides four independent sound channels. The game generates eight different "event" sound effects, plus two "continuous" effects when fliers and smart bombs are active.

The sound channel assignments are (see $78f1):

Effect	Ch1	Ch2	Ch3	Ch4
$01: silo low				X
$02: explosion	X	X		
$04: ABM launch			X	
$08: bonus points	X			
$10: start wave	X			
$20: game over	X	X		
$40: bonus city	X			
$80: can't fire				X
flier				X
smart bomb				X

The sound played when a bonus city is spent is a series of random tones, not a fixed set.

If the slam switch is triggered, the game makes a high-pitched sound on channel 1.

New sounds replace older sounds, though the only actual effect clashes occur on channel 4 because most of the channel 1/2 effects are only used between waves. Firing a new missile or causing a new explosion will reset that effect. The flier / bomb sound will be interrupted by the silo-low warning and the unable-to-fire warning, then resume after they complete. If the flier and a bomb are both present, only the bomb sound will be played.

Odds & Ends

The code has a more structured feel than other video games. For example, subroutines generally have one entry point and one exit point, with code often using `CLV+BVC` to perform an unconditional branch to an `RTS`.

In self-test mode, there are some features not mentioned in the service manual:

- Slam + P1 left fire: draw full-screen grid
- Slam + P1 middle fire: draw color bars, erase screen, then exercise non-MADSEL read/write of video RAM
- Slam + P1 right fire: draw color bars

You can make the trackball more responsive in an upright cabinet by setting the configuration to "mini trak-ball" (R8 switch 4). However, the code (at $5204) only modifies the input if the cabinet is configured as a cocktail setup (R8 switch 8).

If the game ends while player 2 is playing in cocktail mode, during the flashing "THE END" screen, the scores are drawn mirror-imaged at the top of the screen. (Possibly caused by flip flag being enabled at $677f?)

When player 2 is active on a cocktail cabinet, the display needs to be rotated 180 degrees. The hardware does the vertical flip, the software does the horizontal flip, but primarily just for text. The city backdrop, and the position of cities within it, is not flipped. The only time this gets awkward is when drawing the cities, because they use the same orientation-aware bitmap renderer as font glyphs.

The text that scrolls by at the bottom of the screen during attract mode is actually drawn during the initial title explosion sequence. The color palette renders it invisible.

The attract mode game play is not pre-recorded. The computer actually plays the first wave. It targets ICBMs in the order in which they appear, moving the crosshairs to the point the missile will reach in 16 frames. When it reaches the coordinates, it fires an ABM from the closest launcher. There are some limitations, like not having more than two ABMs in flight at once, that slow the pace of the play and make the computer player seem a bit more relaxed toward the end of the wave.

ABM explosions have a maximum radius of 13, but the trackball handler allows the crosshairs to get within 8 pixels of the edge of screen. If you fire a missile near the left edge you can see it peek out on the right edge.

Code at $56d1 appears to be designed to prevent MIRVs from appearing on wave 1, but the test is wrong. (Or not? The comment in the original source code is "# OF WAVE WITH 1ST MIRV", and it's set to 1, so perhaps it's working as intended.)

Unanswered Questions

- Was the screen height originally 232 lines? There are some indications that a line was reclaimed to add 64 bytes of storage.
