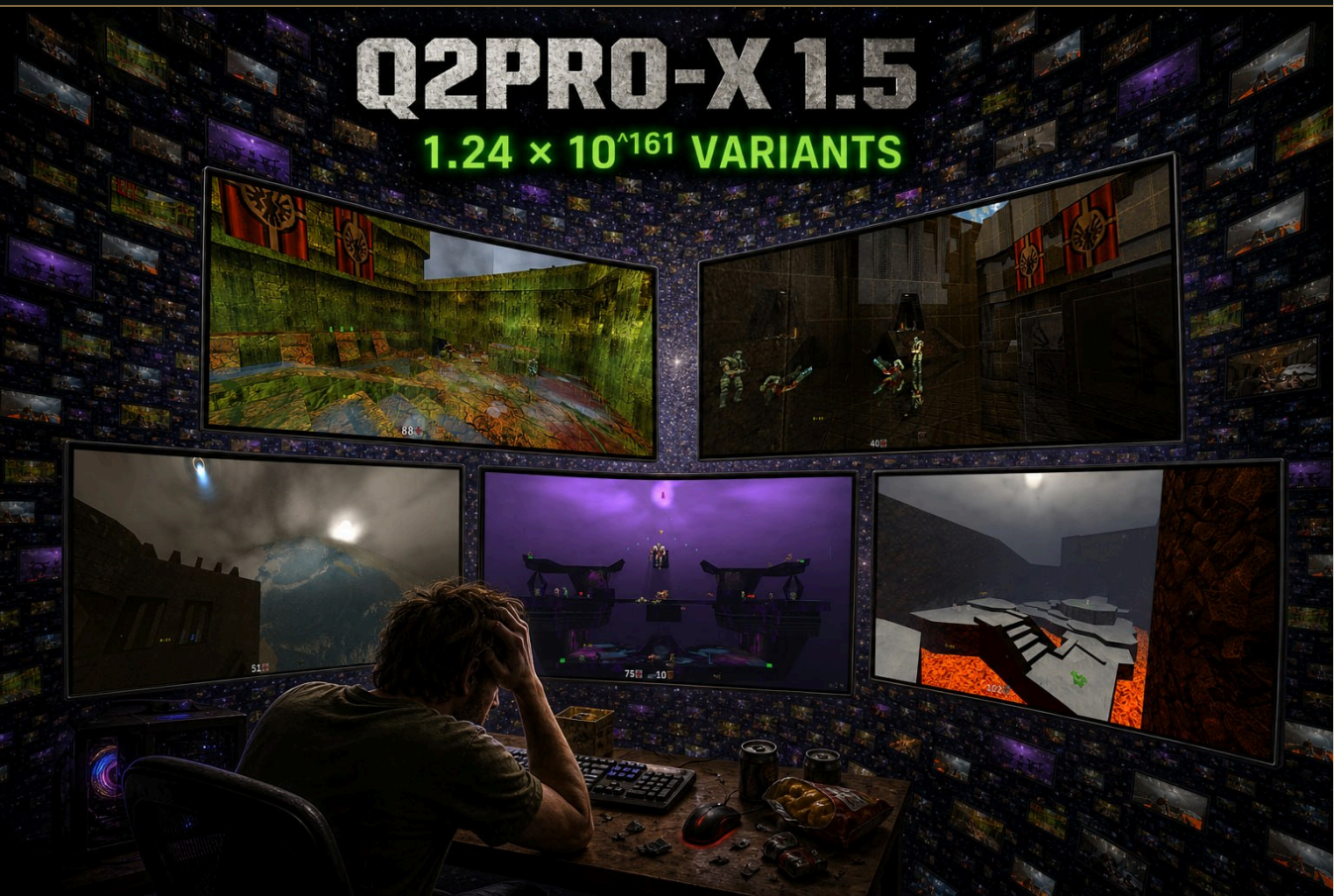


AN ILLUSTRATED CHRONICLE

Q2PRO-X:

History Book



1990-2026

Contents

Prologue

A Shot on a Snow-covered Map	03
------------------------------	----

I. Before Q2PRO-X

Commander Keen, Doom and the Road to Quake	05
Autumn 1997: Quake II Before Release	09
From the First Patch to Tournament Servers	12
R1Q2 and Q2PRO: How Network Play Changed	15
Eraser, Gladiator and Other Opponents at Home	18

II. Developing the Client

Q2PRO-X: The First Betas	20
The First Release and Players' Habits	24
Voice, Menus and Match Recordings	27
Rain on an Old Map	31
Single-player with New Rules	35
The Moon and Stars Above Quake II	39
Earth, a Black Hole and the Map's Inhabitants	44
What Makes an AVFX World	50
The Preset Generator and the Number on the Cover	53
Between the Button Press and the Shot	57
Traces of Battle and the Cost of a Beautiful Frame	64

III. Opponents and New Tools

OpenTDM-X: Teaching a Bot to Win	70
Weapons and Four Personalities	73
Red Armour, Megahealth and Teleports	76
From Studying Demos to Building Tools	79
The First Launch	82

Appendices

Counting the Work Behind the Project	83
What You Can Do in the Game	88

Prologue · May 2026

A Shot on a Snow-covered Map

The snow-covered map had frozen. Footsteps were still audible and the menu opened, but the corridor on the screen would not move. This was how one test build of **Q2PRO-X** behaved in May 2026. It would have been easy to blame a computer struggling with the weather effects, except for one curious detail: sometimes a gunshot brought everything back to life. The same level, the same snow, the same machine—and movement returned. What was the bullet doing that the engine could not do for itself?

By then, the client had spent several weeks being tested with different weapons, old configuration files and window modes. Each fix meant another build and another round of familiar actions. Sometimes a complaint disappeared; sometimes it returned almost word for word. The snow was particularly frustrating: the very image that had taken so much work was getting in the way of playing. Errors turned up in the logs and were corrected, yet the map kept freezing. The answer lay in a detail that was easy to overlook: exactly where the shot that revived it had landed.

The snow was being added to **Quake II**, a game released in 1997. Its maps were known by heart: how long it took to reach the armour, where a lift could be heard, which step let you jump to a weapon. Computers, connections and clients had changed over the years, but players still disappeared around the same familiar corners to escape somebody else's rockets.

Q2PRO-X began with movement and shooting; the weather came later. Before we return to it, we need to go back to 1990, when the future creators of **Quake** were trying to make a picture on an ordinary computer scroll smoothly after a small hero.



Snow cover on a Q2PRO-X map. This later gameplay image illustrates the work on snow that opens this story.

I. Before Q2PRO-X · 1990–1996

Commander Keen, Doom and the Road to Quake

A hero runs to the right and the scenery slides smoothly to the left. In 1990, the owner of a games console would hardly have given this a second thought. On an IBM-compatible PC, that kind of scrolling was still a problem to solve: video memory and redraw speed stood in the way. A small team found a solution, and soon **Commander Keen** was running across the screen—a boy building a spaceship out of household odds and ends. **Apogee** released the game in December 1990. Two months later, its creators had a studio of their own: **id Software**.



Commander Keen. Before the three-dimensional corridors, id Software made platform games. Image from Bethesda's official retrospective.

Keen's success gave the team the chance to make its own games. Several steps still separated it from the shooters to come: Rescue Rover, **Dangerous Dave** in the Haunted Mansion and Hovertank, small projects connected with its work for **Softdisk**. Hovertank already offered a first-person view; Catacomb 3-D added textured walls. The computer was being asked to draw increasingly complicated scenes, but the developers were reluctant to give up speed. Every bit of time saved paid for another detail: a wall, a corner, an enemy ahead.

In **Wolfenstein 3D**, released in 1992, walls followed a strict grid and enemies were flat sprites. On a plan, such a world looks meagre. From inside, it feels rather different: a corridor rushes towards you, a door opens, and the guard behind it is already raising his weapon. Those constraints kept the game fast. The player did not have to pay for free-form geometry—still absent—with a slowdown at the worst possible moment in a fight.

In December 1993, **Doom** made its spaces more varied. Floors and ceilings could have different heights, walls no longer followed Wolfenstein's rectangular grid, and lighting gave rooms distinct moods. What happened outside the single-player campaign proved even more consequential. People connected their computers to play together or fight a deathmatch. A familiar level now had to be studied for a different reason: to work out where an opponent would appear, which route they would take and whether you could intercept them before they reached a weapon. The map became a place to compete, rather than merely a path through monsters placed in advance.



Doom, 1993. Changes in elevation, lighting and network play laid much of the groundwork that Quake and Quake II would develop. Official Bethesda image.

Doom's maps, images and sounds lived in WAD files, separate from the executable. You could play somebody else's level, find an editor and try building your own. **Doom II** expanded on that foundation in 1994; alongside it came Heretic and Hexen, with **id Software** acting as technology partner and publisher. Players grew accustomed to the idea that a game could be remade. Source releases would later widen their options, but it began with accessible assets, editors and the desire to show other people a map of your own.

Quake, in 1996, demanded technical work on a different scale. Levels became fully three-dimensional spaces and polygonal models replaced sprite enemies. You could stand above another player, look up and follow a rocket through the air. The engine had to cull unseen parts of that world, light it and draw it quickly on computers that did not necessarily have a graphics accelerator. Software rendering therefore existed alongside emerging hardware-accelerated versions. The later source release preserved this history in the structure of the project itself: WinQuake, GLQuake and QuakeWorld addressed different sides of the same game.



Quake: three-dimensional levels and models changed both the picture and the ways players could fight. Official Bethesda image.

A fast computer alone was not enough for a network match. A command had to reach the server and its reply had to return. If the picture waited for every exchange, even a step forward felt late. QuakeWorld let the client reproduce its own movement ahead of time, then reconcile it with the server. On screen, the player was already walking while confirmation was still travelling through the network. This solution needed refinement too: release notes record fixes to prediction, steps, frame rates and demo recording. The effort to make online movement responsive began long before **Q2PRO-X**.

I. Before Q2PRO-X · 1997

Autumn 1997: Quake II Before Release

Little remained of its predecessor's medieval castles in **Quake II**. The player arrived on the Strogg's industrial planet: factory interiors, a military landing, heavy music and bodies grafted to machinery. Several maps formed a larger section of the campaign. Completing a task in one place could open a route elsewhere, so a familiar corridor sometimes had to be visited again after something had changed. The name promised a continuation of **Quake**, but the game was choosing its own structure and appearance.

Quake II had to remain recognisably the same game on a machine with a graphics accelerator and one using software rendering. Coloured lighting, transparent surfaces and moving models had to be drawn quickly in both cases. Rendering, network communication and the game module were separate parts of the engine later called id Tech 2. A few years on, that arrangement would allow one part—the renderer or the rules of a match—to be replaced while keeping the rest. At the time, the separation helped assemble a single game from parts that each presented plenty of difficulties of their own.

Q2 Test became available on 19 October 1997. The game was still being made, yet an early version had already left for other people's computers. The development diary for that day records a dilemma familiar to anyone who has prepared a software release: a program changes daily, and the ideal moment—ready to test, with nothing left to fix—never arrives. You have to choose a day and let people try it. A full demo was promised later, after the retail release

was finished. For now, **Q2 Test** offered a first encounter with the future **Quake II** and showed its developers how the game behaved outside the studio.

As the test spread to other machines, the list of fixes kept growing: sound, input, saves, installation and network prediction. Some bugs stopped the program; others let people play but spoiled movement or shooting. The retail release in December carried the number **Quake II** 3.05, and updates followed almost immediately. Remnants of the test could interfere too: an old **Q2 Test** game-module library could prevent startup if the program found it before the new file. Sometimes getting a working game began with sorting out its folder.



The original Quake II console background, taken from the game assets. Its metal emblem and brown palette became part of the familiar look of the game.

Network play was part of **Quake II** from the beginning. In deathmatch, players appeared at different points, collected weapons and returned to the fight after dying. The rules were simple; the weapons soon made their application complicated. A railgun demanded a precise hit and left a pause between shots. A rocket took time to reach its target and could hurt an opponent with its blast. A grenade bounced off walls, while a chaingun needed to spin up and consumed ammunition quickly. Tactics grew out of

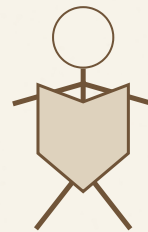
these differences: choosing a favourable range, claiming the upper floor or forcing an opponent into a narrow passage could provide an advantage before the first hit.

A large stack—health and armour—let you survive a mistake, but you had to earn it. Items occupied fixed positions, and armour respawning became an event you could prepare for. This was how the skill of control developed: remember a pickup, estimate its return and prevent an opponent from approaching for free. On screen, it might look like a brief meeting in a corridor. Behind it lay several earlier decisions: where to go after spawning, which weapon to save and from which direction to reveal yourself.

Armour changes an encounter



Unprotected



After collecting armour

The next fight is prepared between shots

Armour changes the preparation for the next encounter. The fight for items begins before the first shot; this illustration explains the principle.

I. Before Q2PRO-X · 1998–2001

From the First Patch to Tournament Servers

The familiar **Quake II** package took shape after the game was released. The multiplayer maps are an easy part of that history to forget: deathmatch was in the December version, but the eight purpose-built levels, q2dm1–q2dm8, arrived in **Quake II** Point Release 3.12 on 19 February 1998. Before that, people played on suitable parts of the campaign and on community maps. The new arenas changed how players encountered **Quake II**. A level no longer had to serve a story objective; it could be built around encounters, weapon distribution and the struggle for advantageous positions.

Patches followed one another. The **Quake II** 3.13 readme warned that old saves would not work and servers predating **Quake II** 3.12 would be incompatible. Behind that short note were the familiar chores of a player: find out what everyone else has installed, locate the right update and preserve the files that matter. Meanwhile, the developers tackled crashes and problems with sound, cooperative play and networking. People had already bought **Quake II** and were playing it while its multiplayer side was still changing.

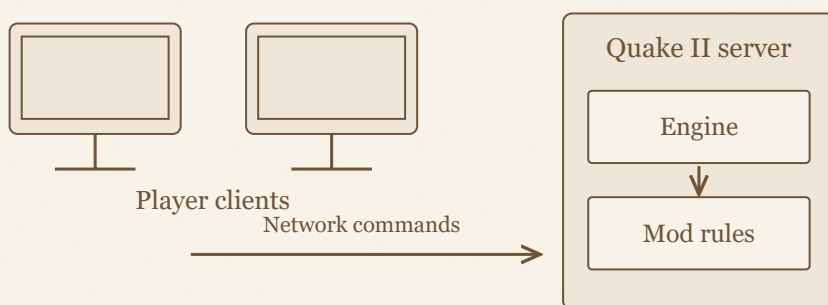
Date	Major releases
October–December 1997	Q2 Test; retail Quake II 3.05; early fixes in Quake II 3.06 and Quake II 3.07.
19 February 1998	Quake II 3.12: eight dedicated DM maps; Q2CTF 1.0 released.
25 February–2 March 1998	Quake II 3.13 and Quake II 3.14; further fixes and CTF updates.
May–July 1998	Quake II 3.15 and Quake II 3.17; continued engine and platform support.
Autumn 1998	The Quake II 3.19 and Quake II 3.20 lines, which became landmarks in the history of the classic game.
December 2001	Engine source code released under the GPL.

In CTF, the familiar map served a different task: bring home the enemy flag and protect your own. Chasing a personal frag could cost the team a win, while knowing a side route and helping a teammate could save the match. **Rocket Arena 2** concentrated on duels in separate arenas. **Action Quake 2** introduced different weapons and its own combat rules. After ordinary deathmatch, players had to learn to judge a situation afresh: the walls and controls were familiar, but the reasons to advance, wait or retreat had changed.

A tournament brought demands absent from a casual firefight. Players needed a warmup, an agreed start, the correct teams and equal conditions. Competitive mods took on those duties, including **OSP Tourney** DM from Orange Smoothie Productions. By spring 1999, it was receiving updates and versions that worked with **Gladiator**. **OpenTDM** would later develop this side of the game further. The client displayed the match; the server mod organised it and enforced its rules.

Mods could change weapons, teams and match rules without rewriting the client: the game-module source was available in **Quake II**'s early years. In December 2001, the engine was released under the GPL as well. The commercial game's maps, models, textures and sounds did not become free as a result. The newly available code did, however, allow people to maintain their own clients and servers, port them to other systems and fix bugs years after **id Software**'s main work was over.

A game that can be changed



New rules on familiar maps

How network play is organised: the engine handles the connection, while the mod sets the rules of the match.

I. Before Q2PRO-X · The 2000s–2026

R1Q2 and Q2PRO: How Network Play Changed

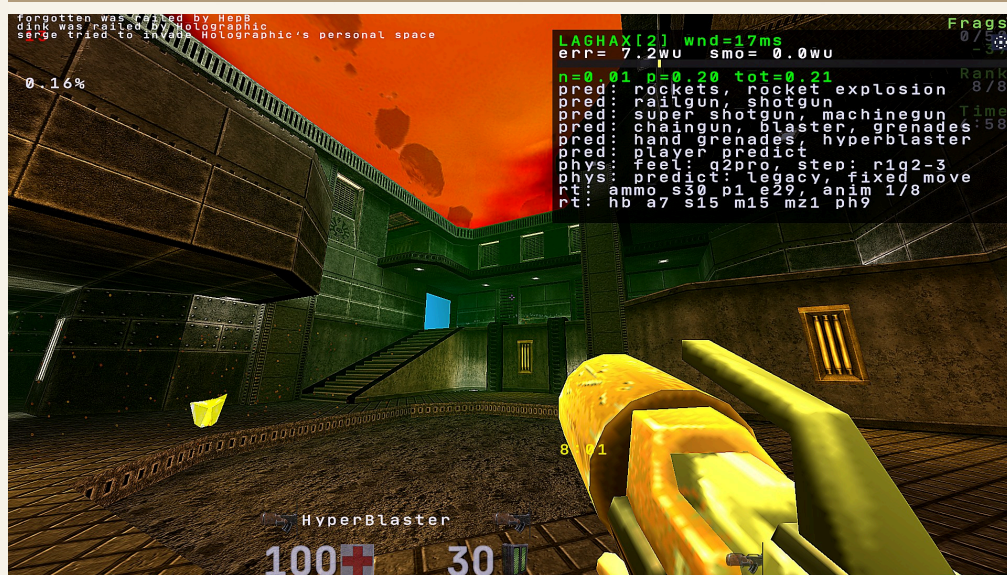
A player might own a new computer and a very old configuration file. They knew how the mouse should move, where the weapon should appear and exactly where to start a familiar jump. Once the source became available, building a custom **Quake II** client was possible; persuading that player to switch was harder. The new executable had to coexist with their maps, mods and settings. Even a useful correction could become a nuisance if a movement they knew suddenly stopped working.

R1Q2 and its renderer, **R1GL**, concentrated on performance and security. The server had to withstand the demands of network play; the client had to avoid unnecessary delay. **EGL**, **AprQ2** and **Q2PRO** developed alongside them, and successful changes travelled between projects. **AprQ2**'s documentation explicitly acknowledges borrowing networking work from **Q2PRO**. Old clients did not vanish on command when a new one appeared: different players chose different versions and continued meeting on the same servers.

Q2PRO also has a history reaching back to the early 2000s: a surviving FAQ thread was opened in January 2004. Even the early descriptions show an interest in MVD, or multiview demos. An ordinary client recording preserves the data stream available to its recorder. A server MVD lets viewers examine a match from different perspectives and follow the contest beyond a single camera. For spectators, it is convenient; for analysis, it is a way to check what an opponent actually had, where they were and why they reached a crucial position first.

Over time, **Q2PRO** accumulated a broad range of networking and user features: multiple protocols, traffic compression, asynchronous client operation, an OpenGL renderer and extensive demo-viewing tools. Asynchronous operation meant that drawing frames and sending commands did not have to run at the same rate. But a high frame rate cannot, on its own, eliminate server or network delay. A good client must coordinate several clocks: input, physical movement, network commands and rendering. A mistake where they meet can feel like a sluggish mouse or a late shot even at high FPS.

Q2PRO became the foundation of **Q2PRO-X**. **R1Q2** remained a useful point of comparison: people brought their settings and expectations from it. When the mouse behaved differently in the two clients, the investigation followed command formation and input delivery. Changing the physics would have been risky—the client could have begun moving differently from the server's calculations. The similar names conceal two distinct relationships: code inherited from **Q2PRO**, and continual comparison with the experience players knew from other clients.



The Laghax diagnostic panel in Q2PRO-X displays prediction and networking parameters during play. This later screenshot illustrates the tools used to examine client responsiveness.



The Movement Basics menu in Q2PRO-X combines R1Q2 mouse behaviour with Q2PRO movement and prediction settings, helping players keep familiar controls.

Quake II found other ways to continue its technical life. Yamagi **Quake II** developed convenient support for the classic game on modern systems, while projects with new renderers explored lighting and imagery. An official enhanced edition arrived in August 2023. None of these developments erased the classic online environment: servers, mods, client configurations and demos still formed their own web of compatibility. **Q2PRO-X** began within that environment in spring 2026. The task was to develop a long-familiar game while allowing people to bring their existing installations and habits, rather than start again.

I. Before Q2PRO-X · The late 1990s and the legacy of early projects

Eraser, Gladiator and Other Opponents at Home

Deathmatch needs another person. In the late 1990s, even finding an opponent did not guarantee an enjoyable game: the connection could be too poor. At home, **Eraser** Bot, CRBot, **Gladiator**, ACE Bot and numerous derivatives came to the rescue. They helped players learn maps, memorise weapon locations and practise their aim. By the time **OpenTDM-X** appeared, computer opponents in **Quake II** had a long history of their own.

Seeing a weapon ahead is not enough for a bot: it must find a way there and decide whether the trip is worthwhile. **Eraser** used route data. For **Gladiator**, BSPC built an AAS navigation system from the map's geometry. Its documentation described personalities, weapon and item selection, observing bots and chat commands; by 1999, additional game modes were supported too. The downloadable game module concealed a substantial collection of decisions, developed further from release to release.

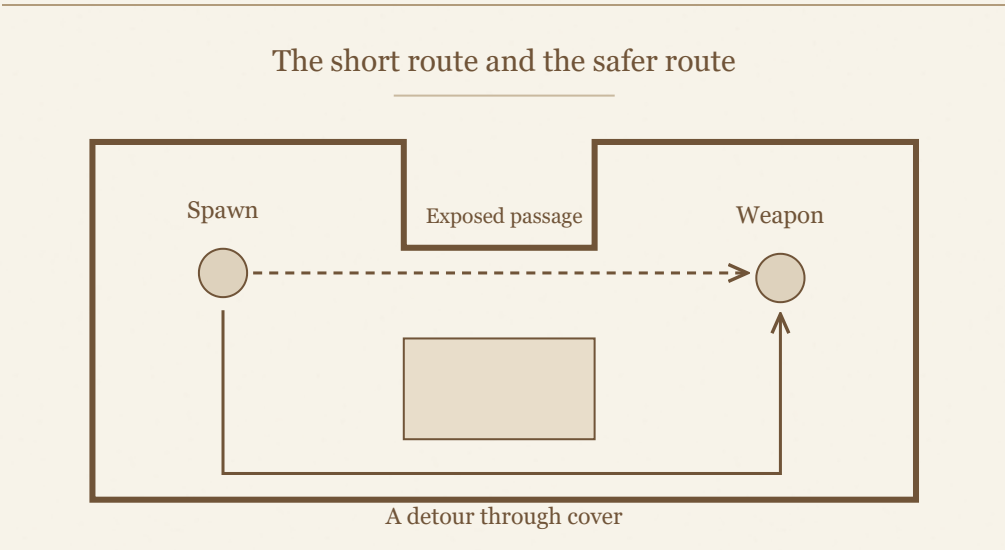
ACE Bot's authors also wanted behaviour closer to a human's. They described client emulation, movement in three dimensions and learning routes travelled by players. A door, a staircase or a jump was supposed to remain a real obstacle for the bot. Learning from people was already an established idea; years later, new projects would have more recordings, computing power and ways to analyse those observations.

Knowing the way does not win a duel, however. A bot can reach a gun first and die immediately by stopping beneath an upper platform. It can aim well yet waste ammunition at the wrong range. Or it can dutifully follow a player

around a loop, giving itself away with every footstep while the player escapes and collects armour. **OpenTDM-X** would eventually face precisely these problems. Correcting one action was relatively quick; coordinating them so that a bot could play a whole match well proved much harder.

That was why real matches and separate movement recordings were collected for the new bot. They showed what people did after spawning, how they escaped an opponent's control and when they committed to finishing a kill. Jumps had to use the mod's ordinary physics; weapon-switching time and rocket flight could not be shortened for the program's convenience. On the server, code and prepared data made the decisions. A local neural model would enter this story later, as a tool for studying recordings outside a match.

That work was still months away. In March 2026, the future **Q2PRO-X** first needed to restore the response players expected: press, turn, shoot. That was where it began.

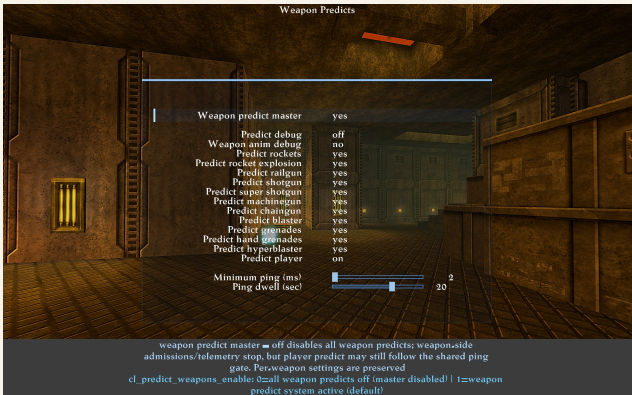


A simplified route choice: a short passage may be more dangerous than a detour. This illustrates a principle, rather than a particular map.

Q2PRO-X: The First Betas

Development of **Q2PRO-X** began on the night leading into 29 March 2026. The foundation was already there: **Q2PRO**, with years of support for online play. The aim was now to see your own shot promptly, find settings more easily and move from another client without unpleasant surprises in the controls. The server still had to decide who won a match. Drawing a shot earlier did not entitle a program to deal damage earlier.

Time passes between a button press and the server's reply. A **Quake II** client already knew how to display its own movement ahead of confirmation; weapons needed several events predicted at once: animation, muzzle flash, sound and trail. A server message describing the same shot would then arrive. Drawing everything again would produce a duplicate; suppressing too much would remove an effect that belonged there. Work began on two sequences of events that had to agree even though one was always ahead of the other.



The Weapon Predicts menu in Q2PRO-X has separate local prediction switches for different weapons. This screenshot shows a later interface for a system whose development began with the first betas.

Q2PRO-X 0.95 and Q2PRO-X 0.95-a

Q2PRO-X 0.95 is the first surviving public release. Its package included the executable, libraries, menus, help and Russian guides. Players added it to their own installations: maps, models and configurations remained theirs. This proved a useful compatibility test. Everything might work in a prepared client folder, only for a setting in a player's configuration to change the behaviour of a weapon or window.

Q2PRO-X 0.95-a followed the first feedback. The blaster behaved differently for a single shot and sustained fire. Hiding the weapon model broke prediction, although its visibility should not have affected shooting at all. Graphics-path selection and Alt+F4 behaviour were adjusted alongside those fixes. This small follow-up release was assembled from very specific annoyances people had encountered in play.

Q2PRO-X 0.96: Familiar Resolutions on a New Screen

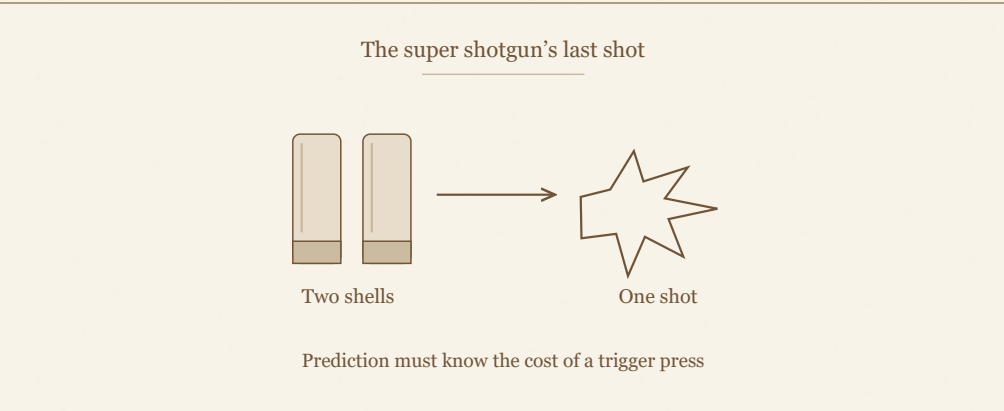
Q2PRO-X 0.96 turned to display modes. Online players often kept a 4:3 aspect ratio and a low resolution on a widescreen monitor. They gained refresh-rate selection, desktop fullscreen with side bars and a separate display-mode switch. The differences became apparent when minimising and restoring the game, so checking the picture once was not enough. A mouse bug surfaced too: the side bars, where there was no game image, were still involved in input processing.

Q2PRO-X 0.97: The Hyperblaster, Lifts and OLED Protection

Q2PRO-X 0.97 examined the hyperblaster: flash, animation and projectile had to belong to the same shot, and releasing the button had to stop the burst. Laghax readings helped trace delay. Lift jitter caused by the interaction between step smoothing and platform movement was corrected. OLED protection also appeared, taking account of client inactivity and black side bars. This short change list covered faults arising in very different circumstances; a single trial launch could not test them all.

Q2PRO-X 0.98: Alt+Enter and Hidden Weapons

Q2PRO-X 0.98 separated the ordinary window from fullscreen modes. Tests covered Alt+Enter, resolution persistence, muting an inactive window and screen protection. Weapon-display settings interfered again: a side model, a centred model, a completely hidden one. Then the observed player was changed. A shot could look right on one installation while an extra action appeared—or a real one disappeared—on another client. Testing returned to both ends of the connection.



The super shotgun uses two shells per shot. Differences like this had to be included when checking the last round.

Q2PRO-X 0.99: The Last Round

Work on **Q2PRO-X** 0.99 began on 13 April. One problem was a shot from an empty weapon: the ammunition was gone, but the client still managed to show a flash. Checking the HUD count was insufficient—the server had not yet reported ammunition spent on shots already predicted by the client. Unconfirmed shots began to be subtracted from the remaining count, and the cost of the next shot was checked. The super shotgun uses two shells; the chaingun's consumption changes as it spins up. The last round forced a careful recount of all the preceding ones.

Some bugs hid from the very logs intended to expose them. One diagnostic mode altered suppression of server effects: turn logging on and the fault disappeared. The logging had to be corrected before its evidence could be trusted. Even then, a match was needed. An automatic check could pass, only for a player to notice another missing or extra shot.

II. Developing the Client · Mid-April 2026 · Q2PRO-X 1.0 · Q2PRO-X 1.1 · Q2PRO-X 1.1A

The First Release and Players' Habits

By **Q2PRO-X** 1.0, movement and weapon prediction shared a ping-adaptation mechanism. The system engaged at the required latency and did not keep flicking on and off when the value hovered around the threshold. Players could nevertheless disable weapon prediction independently of movement prediction. That distinction had to remain consistent in code, menus and hints: a common ping measurement served several settings, each with its own switch.

OpenTDM support also reached the client's team interface. Information came from the server; **Q2PRO-X** could present it more conveniently, but was not supposed to obtain anything the player had not been told. In tournament play, that boundary matters: making team status easier to read is one thing, gaining information unavailable to an opponent is another.

Q2PRO-X 1.1 and Old Railgun Configurations

A user's configuration exposed a problem absent with default settings: the railgun trail became invisible. One kind of network event bypassed the old drawing path; even after that was repaired, the problem persisted with some settings. A zero particle scale was another cause. The previous client had kept a visible trail through a different formula; the new one faithfully multiplied its size by zero. For the player, the result was the same: no visible shot. **Q2PRO-X** 1.0 and the following wave of **Q2PRO-X** 1.1 work restored compatibility for both the ordinary trail and the spiral, testing the actual configurations players supplied.

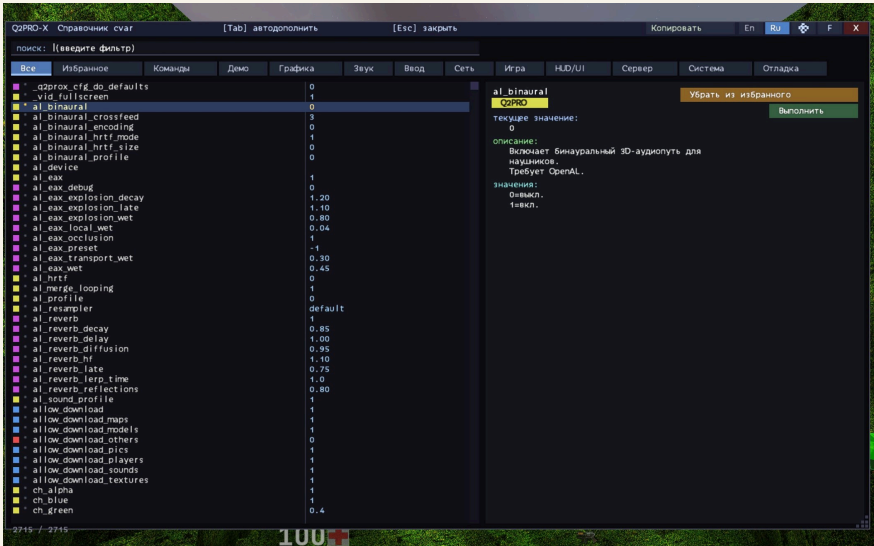
An old configuration often outlived several computers. Its owner had no intention of examining hundreds of lines just because an executable was new. In disputed cases, the team therefore looked at what the setting used to produce on screen. If a small compatibility path restored the expected railgun trail, it was more useful than demanding that every player reconfigure the game. Cases like these gradually built a compatibility layer around the new code.

Q2PRO-X 1.1A: The Mouse, Settings Search and Cyrillic

Q2PRO-X 1.1A continued the comparison with **R1Q2** on 16 April. DirectInput modes were separated, and limited support was added for an old mouse-acceleration fix setting. Reports of unfamiliar movement were traced through command formation and input handling. Altering the physics was tempting: it might immediately feel nicer. But the client could then disagree with the server. Delay and differences in operation order were investigated before changing movement itself.



Checking the familiar railgun trail in early Q2PRO-X versions.



The CVAR browser brought together a setting's name, current value and explanation. It gradually became one of the main ways to understand the client.

The variable browser gained selection, copying, search clearing and a movable heading. The Russian translation of **Q2PRO**'s basic sections was completed. During testing, opening the browser and pressing Alt+Enter caused a crash; repeated switching revealed an interface memory leak as well. Familiar windows then went through the same sequence again: open, change display mode, return, and check what remained on screen and in memory.

A familiar view on a wide monitor



Side bars must not affect the mouse

A 4:3 display on a wide screen. Areas outside the game image must not affect the controls.

II. Developing the Client · Late April 2026 · Q2PRO-X 1.2

Voice, Menus and Match Recordings

Voice chat appeared in the first working **Q2PRO-X** 1.2 builds on 18 April. Until then, a player under pressure had to type, use an external program or stay silent. The client gained microphone selection, push-to-talk and voice activation. Testing began with an ordinary question: could the participants hear each other? Then they changed devices, switched maps and reconnected—and found cases where the same settings no longer produced the same result.

Two connected clients revealed what a single-client experiment could not. Old state survived disconnection, settings updated incorrectly, and some microphones needed more volume range. Device selection, push-to-talk, voice activation, sensitivity and sound levels took shape. At the same time, a large media library had to fit into the user package without unnecessarily inflating the executable. Work continued until setting up a microphone no longer required knowledge of the voice system's internals.



Built-in voice-chat settings. Choosing a device and understanding connection status mattered as much as transmitting sound.

Cyrillic also took several attempts. A character could exist in the font and still be hard to read at the chosen scale, fail to fit its cell or look out of place beside Latin letters. Search input and changing languages inside an open menu were checked separately. A clean configuration concealed some problems: they appeared only after switching display modes or returning from another window.

The server browser gained search, filters and favourites. Players wanted to keep server addresses and familiar configurations even while the interface changed. Local network-port selection was investigated separately: an unsuccessful attempt had to finish within a reasonable time. Otherwise, a player selected a server and stared at a motionless window, unable to tell whether the program was waiting for a reply or had stopped working.



Q2PRO-X's server browser. A list of addresses gradually became a tool for choosing a game and finding it again.

In a dark corridor, an opponent's model could easily merge with the background. Strong highlighting solved that problem at the cost of another: the player became a flat patch of colour. Ways to preserve shape, texture and visibility together were explored. Team membership and the information actually available to the client had to be respected. Highlighting weapons on a map likewise relied on received item data. This work later extended to individual parts of models and newer graphics formats.

In the second half of April, returning to your own matches became easier. The demo library gained search, file information and playback controls. First-person weapons were restored in MVD viewing: server recordings contained different data from what the ordinary viewer expected. The missing view was reconstructed from available player information. The new controls now helped people understand a match, rather than merely skip through it faster.

Search and keyboard handling needed several more passes before those windows were finished. Some characters were rejected, events reached the wrong field and changing recordings could leave the wrong appearance behind. Demo-viewing settings were separated from live play; otherwise, someone could alter a recording's appearance, join a server and discover that their game had changed too. By the end of April, connecting, talking and reviewing matches were more convenient. The next work began with a question that had scarcely belonged to a client before: what would happen if rain fell over an old map?

Статистика игры

Q2PRO-X Статистика игры	
Игрок: [Q2PRO-X]-TECH-test	36500
Сводка	Матчи
Реальные матчи	24
Победы / поражения / ничьи	11 / 13 / 0
Не засчитано без финала	14
Процент побед	45.8%
Фраги	247
Штормов очков	9
Изменение счета	238
Убийства / смерти	122 / 84
K/D	1.45
Зафиксированные сезоны	240
(включая тесты и наблюдение; не входит в матчи)	
Наблюдение	16
Только разминка	10
Не подтверждено	109
Неоднозначная личность	8
Любимая карта	0173
Любимый режим	Дуэль
Источники	DMZ 240 / MVP2 8
Любимое оружие для убийств	Ракетница (152)
Чаще всего убивало тебя	Ракетница (30)
Всего подборок предметов	1550
Rocket Launcher	100
Health	188
Combat Armor	173
Chaingun	150
Armor Shard	119
Super Shotgun	86
Bulldozer	85
Body Armor	74
Shells	59
Rockets	58
Jacket Armor	56
Bulldozer	55
Grenade	51
HyperBlaster	40
Meat Health	40
Machinegun	39
Grenade Launcher	28
Slugs	23
Cells	20
BClock	10
Ammo Pack	9

Esc обновить Ctrl+R пересчитать Ctrl+L ники S сортировка Tab выкладка Esc закрыть

Выполняет действие «открыть статистику игры»
command game stats open

The Q2PRO-X statistics summary brings together results from 24 matches, including wins, losses, frags and item pickups.

II. Developing the Client · 29 April–17 May 2026 · Q2PRO-X 1.3

Rain on an Old Map

Rain appeared in the first builds of the **Q2PRO-X** 1.3 line, begun on 29 April. The new system was named Advanced Visual Effects—AVFX. Nobody rebuilt the maps for it: snow and water entered the same courtyards, corridors and shafts that players had crossed for years. The remaining task was to determine where the sky was, how far a drop would fall and what lay beneath it. Otherwise, rain would pass through ceilings and snow would whiten an exposed staircase and an interior room alike.



Rain and wet surfaces on a Quake II map. This image from a later guide shows the development of the system begun in Q2PRO-X 1.3.

A drop made rings on water and broke into splashes on stone. Floors darkened, puddles formed and reflections appeared in them. Individual effects could be adjusted fairly quickly; together, they sometimes looked as though they belonged to different games. Get transparency wrong and water became paint. Get drawing order wrong and a reflection sat on top of the floor like a separate picture. What needed adjustment was an entire wet scene: its lighting, surfaces and everything that landed on them.

Water ran to the edge of an upper platform and fell below, gradually building another puddle. In a May test, that lower puddle behaved oddly: looking away made it disappear; turning back made it appear full at once. The cause lay in its relationship with the stream. The program created the water while drawing the stream and forgot it when there was no reason to draw. Accumulation was separated from visibility. The platform could now keep getting wet even after the player entered another room.

This memory needed rules of its own. A source can stop feeding a stream without the water below disappearing that second. A camera can enter another room without a puddle ceasing to develop. Yet there is no sense in continually drawing an invisible mark. State and image were therefore separated: one keeps track of how much water arrived and what happens to it; the other decides what the current frame should show. The same distinction would later help with blood, reflections and autumn-scene optimisation. Watching rain was uncovering general rules for building the world.

Water also demanded a sense of scale. A large surface receives rain rings and light; a small puddle may show a footstep, bubble or brief splash. Lava and other liquids have their own appearance and must not accidentally inherit water's tint. Fog, spray, foam and a wet lens belonged to the same scene, but did not have to be enabled together. Players could retain a useful layer and remove a distracting one. Controls for particle and effect families developed alongside their appearance, and master switches were checked for work continuing invisibly after shutdown.

Snow made the old map's seams particularly obvious. Coverage calculated separately for each polygon exposed its boundaries; a continuous white plane, conversely, passed through steps. Even footprints needed attention at first: random dents do not look like steps with a left and a right foot. Testers crossed the snowy courtyard again, inspected the edges and returned to the surface calculations.



Snow exposed seams, steps and the way a covering met the map. Adjusting particle count alone could not fix them.

The first approach to volumetric snow led nowhere. Joining sparse particle-impact points into a continuous surface produced floating plates, stripes on stairs and snow in unsuitable rooms. An unknown cell in the field was mistakenly treated as existing floor. That approach was stopped in favour of what was actually known about the map: where a surface existed, whether precipitation could reach it and how it sloped. Research compared GPU snow-accumulation approaches, including published graphics examples. The GPU took on the covering's pattern, relief, sheen and soft edges; geometry remained the basis for deciding where snow was allowed to exist at all.

Moving close to a snowdrift quickly removed any leniency. A light patch might suffice from afar; nearby, its height above a step, the depth of a footprint and the shadow on a slope became visible. Different snow materials and lighting from the Sun, Moon and lamps followed. A patch adjusted in daylight was reopened at night: pleasing whiteness could become an overexposed blotch. A tester walked the map, turned and rose above the surface—and found edges absent from the screenshot chosen for the report.

In May, the snowy world began to freeze—the incident with which this book opened. Two real sources of wasted work were found first: on maps without water, the program searched for water too often, and checks for open sky repeated unnecessarily. Both were fixed. Snow became cheaper to calculate, but the picture still stopped. Those bugs had existed alongside the main fault, rather than explained it. An average frame-rate counter offered little help: between freezes the game ran well; during them, the three-dimensional world stopped updating while menus and sound carried on.

A shot into a surface where snow had already accumulated supplied the answer. Its mark and particles appeared—and the world revived. Drawing them restored OpenGL to the state the engine expected. The new snow pass had changed graphics settings directly, bypassing the engine's internal tracking; the next frame began on the false assumption that everything was already configured correctly. The shot happened to repair that disagreement. Settings were then restored after the snow pass and the engine's tracking brought into agreement. Repeat tests on `ztn2dm3` and `q2dm8` confirmed the fix on 17 May.

Less visible work continued during the hunt for the freezes: removing repeated highlight calculations, comparing particle costs and adjusting graphics-pass order. The faster version was not always kept. Frame comparisons could reveal a changed tint, forcing a return to the previous blending and a search for savings elsewhere. Some measurements were wasted: a different bloom setting or frame limiter had remained in one configuration. The whole run then had to be repeated with identical settings. Such days added no line to the client's feature list.

Q2PRO-X 1.3 was published on 17 May. Precipitation now accumulated on old maps, water flowed between platforms and snow retained footprints. Puddles reflected the sky, though it was not yet as detailed as intended. The following weeks were spent giving those wet courtyards a recognisable Moon, stars and changing light overhead.

II. Developing the Client · May–August 2026 · Experiments for Q2PRO-X 1.5

Single-player with New Rules

On a familiar arena, it is easy to hear a shot before seeing who fired it. But if a Strogg comes around the corner instead of another player, habit no longer supplies every answer. It has a different weapon and pace of attack; a passage well suited to a duel can become a trap against several monsters. Experiments with this kind of single-player game began in May, after **Q2PRO-X** 1.3. By **Q2PRO-X** 1.5, they formed a separate part of the client, with their own rules, settings and game library.

Why a Separate Game DLL Was Needed

Many earlier changes could be shown without interfering with the server's decisions: drawing snow, lighting a wall or processing sound differently. A picture cannot decide how much damage lava deals or where a new monster appears, however. In **Quake II**, those duties belong to the game library, the game DLL. **Q2PRO-X** received its own version in a separate folder. The ordinary baseq2 library stayed in place, and the new rules applied to local single-player. They did not travel with a connecting client onto somebody else's server.

The first idea for shaking up the campaign was simple: stop knowing in advance what waits behind a door. A mutator changed the arrangement of items and monsters. Conservative settings kept opponents at roughly the same strength; a freer mode allowed neighbouring tiers, and a chaotic mode made encounters less predictable.

Keys and items required to finish a level were protected. Without that safeguard, novelty would end at the first locked door for which no key remained.

Randomness could be saved too. A zero seed generated a new arrangement on loading, while a specified number allowed it to be repeated. That helped both play and debugging: an unfortunate case no longer vanished on the next launch. Replacements still had to respect the level itself. An aquatic monster stranded on a dry platform did not make the campaign more interesting; it merely exposed a programming mistake.

Monsters on Deathmatch Maps

A multiplayer map already has spawn points, weapons, armour and corridors built for combat. Training mode used that foundation: monsters appeared on suitable DM maps even though the game was being played alone. Players could choose their number and difficulty, allow them to respawn and set the delay. A new creature was placed at an unoccupied point; a position immediately beside the player or another living monster did not count as suitable.

Weapons, health, armour, ammunition and power-ups could respawn in their original places. Training therefore did not become a one-off clearance followed by a map with nothing useful left. The player returned for supplies and began to judge familiar routes differently: this lift offers an escape, that corner is a good place to meet a pursuer, and here is somewhere you must not run out of ammunition.

An optional scored round gave the session a measurable goal. Kills earned points, damage received lost them; reaching the chosen total meant victory, while the player's death meant defeat. This was a separate branch of single-player, rather than an early version of **OpenTDM-X**'s duel bots. Strogg and trained opponents solve different problems and follow different logic.

Water You Can No Longer Jump Into

Liquid replacement tested a player's memory even more sharply. On the ordinary map, they might cut through water without thinking; after replacement, real lava lay there. More than its colour changed: the game library applied the new liquid's damage and physics, while visuals and AVFX contacts followed the same substitution. The reverse option turned lava into water. Both directions could be enabled together: each original body of liquid changed independently, without an endless sequence of conversions back and forth.

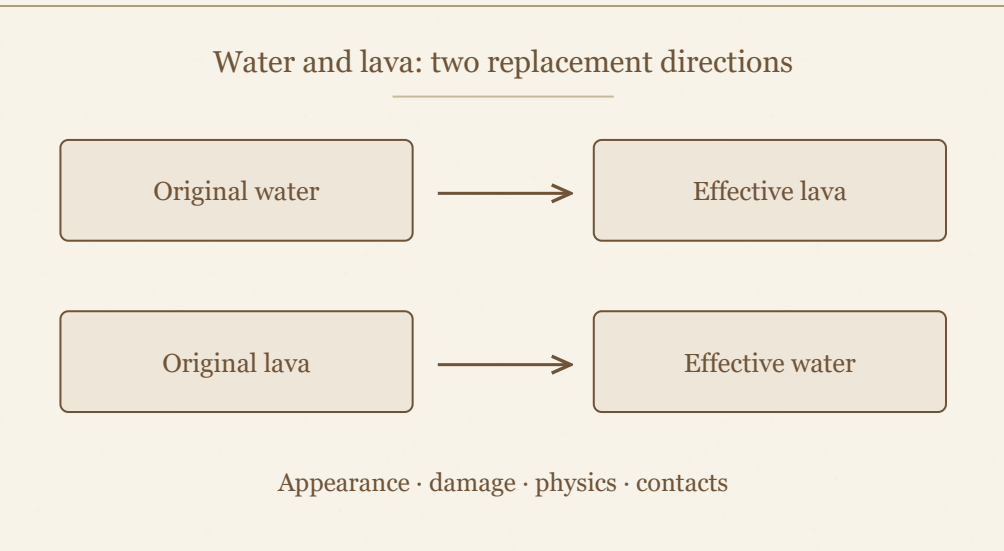
This was an easy place to create a beautiful but incorrect scene. If water already burned while snow and rain still treated it as cool, the player saw two incompatible accounts of what was happening. Material, damage and contact effects had to be connected. These experiments were limited to single-player; in a network match, a client has no right to turn a safe pool into a hazard on its own.

Familiar Walls, Unfamiliar Silhouettes

Another set of experiments changed the level's visible shape: surfaces gained slopes and irregularities, and suitable sharp corners were rounded. AVFX could follow the altered visible surface so that a puddle or footprint did not float separately from the floor. This changed the image, not the map used for movement: collision retained the original geometry. That distinction matters, and must not be confused with actual liquid replacement or the hardware tessellation developed in the renderer.

Repeated monsters were made visually distinct too. Scale, colour, proportions and asymmetry could vary while hitboxes, damage and behaviour remained unchanged; feet had to stay on the floor. Some other local settings deliberately gave AVFX a gameplay effect: a meteor could hurt a monster or explode a barrel, and lightning could strike a destructible object. Such effects were disabled by default. A handsome sky was not supposed to alter campaign difficulty without permission.

Free World and the ability to watch wildlife arrived later. The important shift had happened earlier: **Q2PRO-X** was no longer only a different way of looking at **Quake II**. Within its own single-player game, it let people change the experience itself while keeping the familiar version available alongside those experiments.

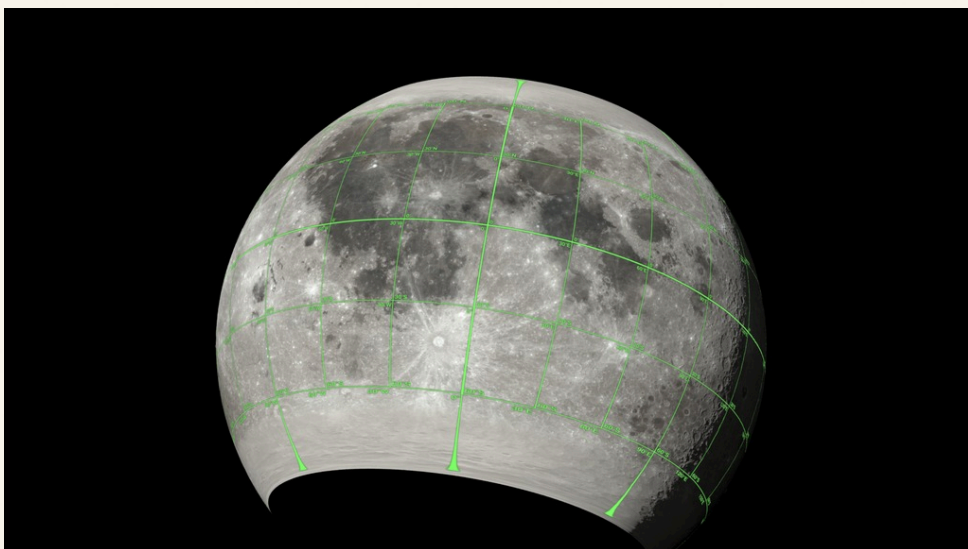


In the single-player experiment, the effective liquid changes: appearance, damage, physics and AVFX contacts. The diagram shows two independent replacement directions.

II. Developing the Client · 18 May–early July 2026 · Q2PRO-X 1.4

The Moon and Stars Above Quake II

The sky occupied almost the entire **Q2PRO-X** 1.4 development line. It appeared through courtyards, windows and openings, sometimes as no more than a narrow strip above a wall. Clouds and celestial bodies were positioned with those gaps in mind. Some maps still used the old skybox: in June, one level was found where sliders changed their values while the sky remained unchanged. The new graphics path worked, as did the settings; the map simply never reached it.



How a flat map becomes the Moon's surface. NASA Scientific Visualization Studio's illustration for its CGI Moon Kit, based on LRO data. This is explanatory NASA material, not a game screenshot.

A Moon Worth Looking At Closely

At first, the Moon's seas and craters were invented procedurally. The pattern sufficed for a small disc. When the Moon was enlarged, there was a desire to recognise familiar features, rather than merely see relief. A NASA/LROC albedo map supplied the reflectance of the real surface. An 8K HD option followed. Details worth consulting lunar imagery for became distinguishable on the large Moon, while lighting preserved its phase and spherical form.

The large Moon developed peculiarities of its own. A clipped copy sometimes appeared elsewhere in the sky, stars shone through its dark half, and brightening the crescent could brighten the whole night. The disc's light, the surrounding background and illumination of the map had to be adjusted separately. Otherwise, improving the Moon inevitably spoiled something beside it.

Stars from a Catalogue

Stretching a handsome photograph over the sky was not enough. As scale changes, bright points on a texture become blobs, and recognisable constellations can easily end up in the wrong place. The raster-atlas approach for point stars was rejected in late May. Catalogue data was prepared instead: stars received individual coordinates, and constellation lines used real reference points. Hipparcos identifiers supported cross-checking. The Milky Way followed the orientation of the galactic plane, and planets were separated from the fixed stellar layer.

A background layer remained between the catalogue stars. It filled the sky without moving recognisable constellations or changing their pattern for decorative twinkling. Large open maps revealed a mirrored repetition below the horizon. An unfamiliar spark could escape notice; a second planet or a repeated group of familiar stars immediately exposed the error.

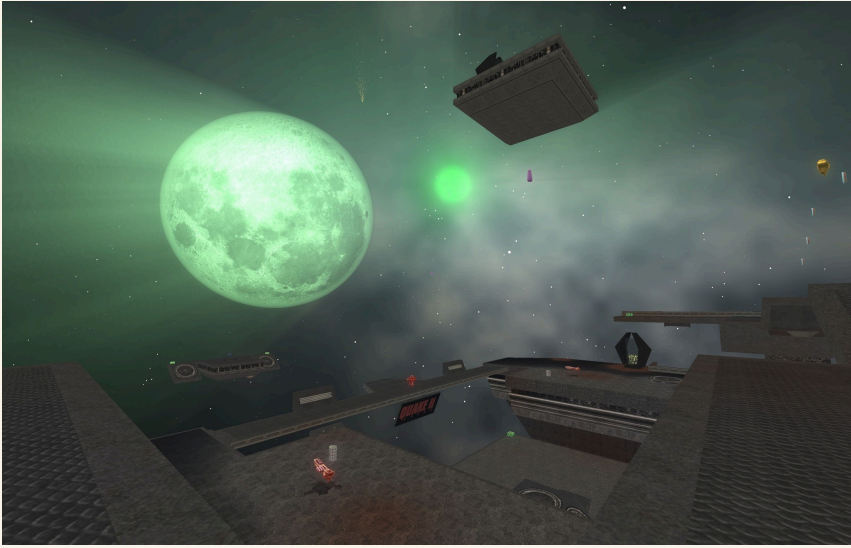
The data was checked outside the game renderer first. An overview was drawn, bright reference stars and constellation segments were verified, and unnecessary bulk was removed from the original catalogue. Several ways of supplying the result to the GPU were then needed. The modern graphics path held more objects; the compatible path held fewer. An expensive graphics card did not guarantee that the chosen API exposed the required feature. The client therefore reported which path it was actually using. Users could understand a limit on their installation instead of fruitlessly moving the star-count slider.

Meteors and comets moved, leaving tails, light and smoke. Clouds could obscure them, and impacts left separate marks on surfaces. Sunbeams, eclipses and coloured moonlight changed floors, water and models. Testing meant looking down as well as up: a celestial event could be correct above the walls and quite different in a puddle underfoot.

Water gained waves, foam and reflections. Old maps did not tell the new renderer how to place them: their surface and material data had been designed for the earlier game. A successful setting was therefore taken to another level, where a seam, wrong direction or unsuitable edge often appeared. Then the first map had to be revisited to make sure the correction had not spoiled it.

A Wider Range of Brightness

HDR required brightness to survive the entire frame pipeline. Intermediate images, bloom, exposure and the final transformation lay between the original light and the monitor. If one stage clamped values too soon, later processing could not restore the lost information. Higher-precision buffers and a corresponding output path were used. From outside, faults looked very different: a black screen, an excessively bright night or a bright object standing out incorrectly. Each had to be investigated stage by stage, checking what the next part of the renderer received.



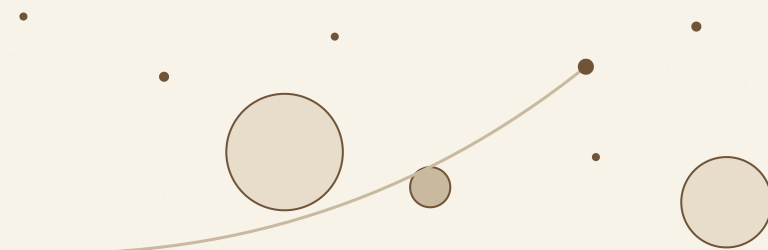
A gameplay scene from the HDR testing materials: a bright Moon and luminous sky above a dark map. This ordinary screenshot shows the composition, but cannot reproduce the full brightness range of an HDR display.

One bright sky effect had to be rolled back when unwanted geometry appeared in the frame. Gamma was corrected separately in night scenes, followed by checks of the wet-lens effect, which processed the finished image. Screenshots brought another difficulty. An HDR file opened in an unsuitable program looked different from the game; the viewer had to be checked before the image could be used as evidence of a fault. JXR saving and white-level handling were added for those captures.

Local play expanded during the same weeks. Players could visit a familiar deathmatch map with monsters, practise or simply examine the scenery. Competitive configurations were kept separate from the appearance of such an outing. Otherwise, after an evening of weather and rich effects, a player might join a server with settings that made an opponent harder to see.

These developments became part of **Q2PRO-X** 1.4 by early July. Checking a shot now took longer than in spring: beyond the weapon itself, there was light, reflection, a wet lens and the result of switching graphics paths. The collection of familiar test maps grew without discarding the old ones. More celestial objects and materials were already being prepared for the summer release.

The sky is made of independent layers



Star background · celestial bodies · movement
Each group has separate settings

The sky's background, individual bodies and moving objects form separate layers. This diagram is not to scale.

II. Developing the Client · July–August 2026 · Q2PRO-X 1.5

Earth, a Black Hole and the Map's Inhabitants

Earth appeared beside the Moon in early July. Textures were prepared at 4K, 8K and 16K: the difference was barely visible on a distant disc, but enlarging it revealed detail across the continents. The planet rotated and remained an independent object. It could be seen alongside the Moon and Sun, so the old approach of merely replacing one sky picture with another no longer sufficed.

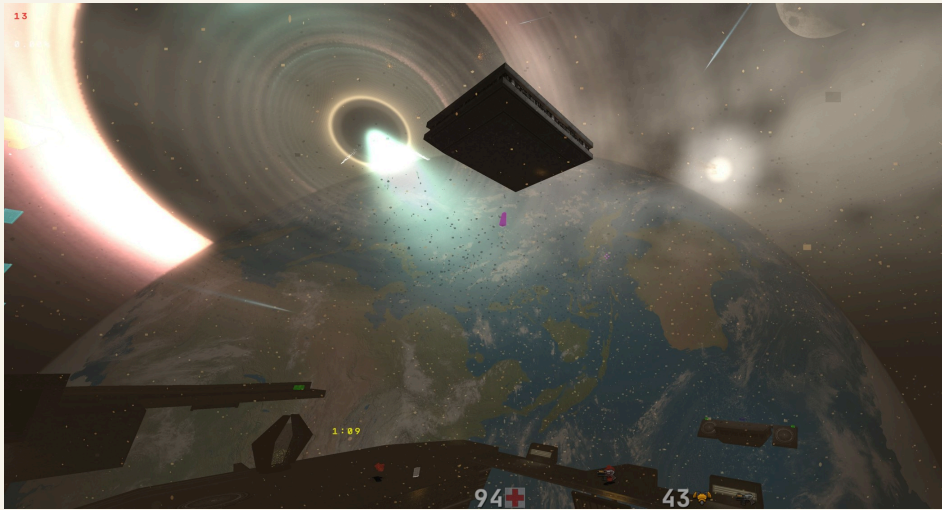
Testing quickly moved beyond the planet itself. If Earth covered the Sun, the solar disc could not remain drawn over it. The same applied to bloom, volumetric light and the final screen-space rays. Each layer took its own route through the renderer, so a correct shadow at one point did not guarantee a correct frame. Too broad a mask removed useful light along with the excess; too narrow a mask left a glow on the planet's surface. Several revisions were needed to reconcile the visible edge with the soft pattern of rays around it.

Light Around a Black Hole

The next object required a look at NASA visualisations. Around a black hole, the far side of the accretion disc seems to appear above the dark centre: light bends around it. The near and far sides differ in brightness because the hot material is moving, and narrow images of the disc appear close to the shadow. These features were studied in scientific materials. Drawing a fiery ring was

easy; producing the recognisable arrangement in which part of the disc passes in front of the shadow and another part appears above it was much harder.

Existing graphics examples proved too expensive. One repeatedly traced a ray before calculating the disc and noise; across a full-screen image, the step count reached billions. Such a picture might be viewed on its own, but not used as the sky over a playable match. Scientific material supplied visual reference points, while the game's calculations were bounded. The result was a visual model of characteristic light distortion, rather than a complete simulation of general relativity.



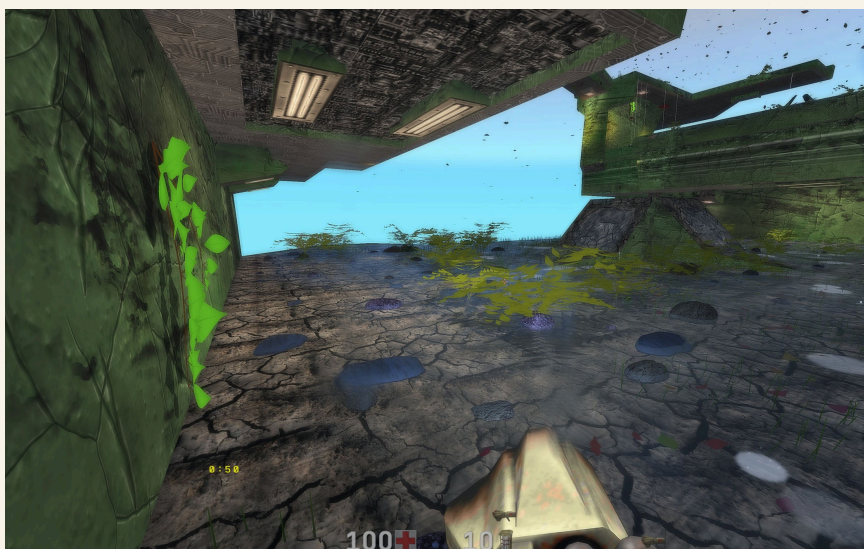
The celestial scenery brought real astronomical data together with graphical approximations for a game. The black hole had to work with the sky, map and reflections.

The hardest part came when the existing sky entered the distorted region. A star had to move, be covered by the shadow or disappear; its old copy could not remain with another drawn on top. Earth, the Moon, planets, clouds and luminous objects were checked separately. Water and puddles added another view of the same scene. If a reflection drew the sky differently, the black hole could become two different objects in the main view and underfoot. Distortion, shadow and emitted light therefore had to agree across several passes.

Later, the black hole could capture matter: meteors and comets changed course, stretched and vanished, while solar plasma formed streams. A mundane bug emerged from the spectacle. If the black hole itself moved, a captured object could chase it forever. Capture duration had to be bounded. Passing spacecraft added further checks: their appearance, reflection and sound had to fit everything else happening in the sky.

Materials and Vegetation

Walls and floors were changing meanwhile. Original textures gained three-dimensional details, cracks, plants and seasonal materials. Geometry determined where they belonged: the old map authors had left no instructions about where future grass might grow. Mistakes needed no diagnostic numbers—a plant protruded from a wall or floated over a step. Every placement change meant another walk through familiar levels.



Familiar Quake II architecture with new surroundings. Map geometry remains the foundation; extra materials and vegetation have to fit it.

In August, a tester disabled materials, loaded a map and turned them back on. They did not appear. Four more fixes followed the first: real faults disappeared, yet that sequence still failed. Eventually, the investigation reached the surfaces' auxiliary data. Disabling the system freed it, but enabling it did not recreate it. Start the test with materials already on and the client seemed perfectly healthy. After the discovery, testing followed the exact failing sequence: load with materials off, then enable them on the running map.

Wildlife and Snakes

The map's creatures gained turns, pauses, visibility rules and interaction with their surroundings. Eighteen independent switches affected their behaviour—262,144 possible states. Automated runs exercised millions of transitions and found settings being lost. Yet a snake could survive every switch correctly and still move unconvincingly. That required ordinary observation: watching how it chose a route and went around an obstacle.

Free World let people linger in a map without immediately looking for an opponent. They could change the environment, watch its inhabitants and explore the geometry; training with monsters on DM levels developed alongside it. Settings could strip that richness back for a network duel. Movement at the edge of the screen is interesting during a walk, but easily mistaken for an opponent during a match.

Allowing snakes to attack exposed the boundary between two parts of the game. A decorative animal exists on the client; the server calculates health and damage. A link to a game entity was added for optional snake combat in local single-player. Collisions checked that it was still the same entity: a freed number might already belong to another object. On a remote server, a decorative snake gained no power to interfere with the match.



Local modes offered different uses for old maps and monsters. Their gameplay rules were kept separate from the client-side effects of online play.

Even a snake's death mark exposed quirks of the old renderer. If it died beneath opaque water, a stain on the bottom could not be seen. Blood had to reach the water surface, fit its boundaries and move with its waves. Transparent and opaque water were drawn at different stages but remained the same medium. Surface detection was therefore tied to the real water volume, not the picture's transparency. In a narrow pool, the mark needed clipping at the banks so it did not spill onto nearby floor. These cases later helped with the general contact-effects system.

Why Loading Took Minutes

All these changes increased the number of graphics programs needed before drawing. On one output path, compiling shaders from cold took about 153 seconds. Initial changes reduced that to two minutes—still far too long. The large set had to be divided into families, account for different output buffers and prepare the required combinations in advance. After the rework, the measured scenario took about forty seconds on first launch and half a second with a populated cache.

A repeat launch with the cache initially looked fast, but the game then stopped for another six and a half seconds. A forgotten variant without bloom was being compiled on first use, after the map had appeared. From then on, timing did not stop at the end of loading. Testing continued through play and settings changes, while lighting, materials and reflections were compared with their previous appearance.

The public **Q2PRO-X** 1.5 package arrived on 25 August, with 22 Russian and 22 English guides. It was checked in a clean folder. A working installation could conceal a missing asset, an accidentally saved setting or an incomplete preset; a new user had none of those reserves. Even a catalogue containing hundreds of variables could be silently truncated. Once the package had been checked and published, work moved to **Q2PRO-X** 1.6.

II. Developing the Client · Summer 2026 · Q2PRO-X 1.5 features

What Makes an AVFX World

On a winter map, snow catches the eye first. Then a wet streak below a ledge, steam and the reflection of a distant light. Calling all of it ‘improved graphics’ is convenient until you want to keep one thing and remove another. By **Q2PRO-X** 1.5, AVFX covered several independent families. They could work together, but each had its own job, cost and reason for existing.

Weather and the Air Between Walls

Rain, snow, ash, dust and haze established the air's character. A secondary component could complicate it without replacing it altogether. Storms added lightning, illumination and thunder; a wet lens suggested looking through a picture covered in drops. These effects needed different placement rules: outdoor precipitation, a roof overhead and an underwater camera could not all be served by one unconditional stream of particles.

A flat sky and a cloud with light passing through it solve different problems. Volumetric fog and clouds gave the air depth; rays revealed the direction of illumination. Simpler sky options remained alongside them: weather, blue, starry, space, nebula, sunny and mystical skies. Celestial bodies and moving objects were chosen separately: a planet could stay in place while a comet or ship passed above it. The Moon and black hole belonged to this larger arrangement.

Water Has More Than One Kind of Reflection

A raindrop ripple, a wave, a glint and a reflection do not replace one another. **Q2PRO-X** developed large bodies of water separately from rain puddles that filled, drained and dried. Model reflections made players, weapons and items visible in them. Screen-space scene reflections reused the visible image of the world. True mirror surfaces formed another system: a floor, slope or wall could reflect its surroundings from a different direction.

That was why ‘reflections on’ said little about the cost of a frame. One scene needed only a watery glint; another required drawing the world and models again for a mirror. Settings separated reflected-object categories and quality. Users could keep the part that made them choose a look in the first place, while performance tests could better identify the GPU's work.

A Wall Acquires Relief

At first, a new surface may look like another texture. Approach from the side and the difference becomes clear: masonry projects outward, stone is uneven, and the floor's edge is no longer perfectly flat. Hardware tessellation divides a surface into smaller patches from which detailed relief can be formed. In **Q2PRO-X**, it covered architecture and liquids, including water and lava. This path used Native WGL; EGL retained ordinary world rendering. A checkbox could not erase the difference between the two graphics backends.

Tessellation was not merely an extension of the single-player wall-deformation experiment: it could also be used online. But visible relief did not mean the server had acquired new steps or cover. Added brickwork and grass were no reason to change a player's path either. Keeping a beautiful surface connected to the original playable map proved just as important as giving it depth.

Materials were built in layers: mould, water running down walls, greenery, cracks, brickwork and floor or wall coverings. Boards, stone and tiles changed a room's character; their order and coverage determined whether it remained recognisable. Surface objects, three-dimensional grass and wildlife formed neighbouring families. They should not become appendages of one common switch: someone might want an overgrown wall without snakes, or a clean floor with a few decorative details.

Leaving Your Own Mark on a Map

Some scenery could be made as well as selected. Drawing tools placed images on map surfaces; a separate system placed volumetric steam and smoke. Libraries of drawing materials and steam shapes made those assets reusable. This meant working with particular places: smoke rising from this crack, an image remaining on that wall. A random generator was not supposed to erase such placement at the next button press.

Flashlight illumination, monster contacts and the sounds of rain, thunder, footsteps and steam tied the families to nearby events. A shot into snow, water or a hot surface could no longer be represented by the same flash. That requirement later grew into the extensive **Q2PRO-X** 1.6 contact-effects work: a mark needed the right material, location and lifetime, not merely a convincing colour.

All these features still had a place in ordinary game settings. AVFX could be enabled for demo viewing and restricted differently in an online match; a competitive mode could be selected and expensive families removed. One person wanted to film a snowy arena, another to recognise an opponent's silhouette immediately. They did not have to agree on one mandatory appearance for **Quake II**.

II. Developing the Client · August 2026 · Q2PRO-X 1.5

The Preset Generator and the Number on the Cover

On this book's cover, a person sits before many screens showing **Q2PRO-X**. On one, greenery covers a familiar arena; on another, the sky darkens overhead. Light, surfaces and air change from screen to screen. Above them is an enormous number. The picture has a particular history: in August 2026, a question arose about how many visual combinations the client already permitted. The answer proved far more interesting than a long list of effects.

One Map, Thousands of Decisions

Imagine setting up an ordinary scene. First we choose snow instead of rain, then a sky, a light colour and wall materials. Each decision seems small, but the next applies to all the combinations already made. By hand, we might find a few favourite looks and stop there. The AVFX preset generator offered a different way to work: describe the kind of scene you want, press Generate and see a combination you might never have thought of yourself.

Users chose the main atmosphere, secondary airborne component, sky, celestial bodies, meteors and ships, atmospheric effects, surfaces, water and puddles. Any area could be fixed or left to chance. Materials and wildlife had separate choices: random selection, forced on or forced off. Full surface coverage was independent too. The generator could thus surprise people in the details without contradicting their main wish—for example, a map without animals.



The AVFX preset generator in Q2PRO-X 1.5. Choices define the conditions for a scene; the scene changes when Generate is pressed.

Smart Chaos kept combinations more familiar. Full Chaos also varied the shape, colour and movement of active families within defined ranges. That did not permit every possible set of values. With rain disabled, requiring rain puddles needed reconciliation; lightning could not silently turn a fixed snowfall into a thunderstorm either. Incompatible requests were explained before a scene was applied.

Six quality levels limited complexity. Volumetric clouds and space skies were not available at the lightest level; model reflections began at Ultra Quality and mirrors at Extra Quality. These were limits on effect selection and cost, not a promise of identical FPS on every computer and map. A generation seed and short summary made it possible to return to a discovery; the result could be saved as an ordinary user preset.

The Number on the Cover

The calculation dated 21 August concerned **Q2PRO-X 1.5**. It examined 2,224 unique control variables and identified 482 independent visual axes in twenty categories. Numeric sliders—for example, an arbitrary raindrop size—were excluded from the finite count. That alone made the exercise more conservative than multiplying every possible menu value together.

It was especially important not to count what was absent from the screen. When the AVFX master switch is off, all the variations of its child effects do not become different scenes: they are all off. The same reasoning was applied to dependent families, light sources and reflections. Ready-made presets did not multiply the result either: they select settings already counted, rather than adding another visual dimension. At that point, 82 supplied visual presets were counted separately.

After dependencies were accounted for, the model produced approximately 1.24×10^{161} graphical configurations—a 162-digit number. That is the number on the cover. It describes the whole examined space of client graphics settings, not the number of presets the random generator is guaranteed to produce. Nor does it prove that every pair of frames looks different: on a map without suitable water, some water settings will simply have no visible effect.

Consider what examining them would mean in ordinary actions. Spending ten seconds on each variant without a break would take about 3.92×10^{154} years. The calculation compared that with an assumed age of the Universe of 13.8 billion years: roughly 2.84×10^{144} such lifetimes. Even involving all 8.2 billion people in the population estimate used at the time barely changed the practical conclusion. The viewing could never be finished.

A large number did not remove the need for checking. The study went through three revisions as master switches and black-hole reflection dependencies were clarified. The result was accepted as an illustrative approximation of diversity, rather than a mathematical proof of the number

of unique images. User drawings and textures remained outside the finite figure. New **Q2PRO-X** 1.6 surfaces were absent from the August calculation too: it is a historical snapshot of **Q2PRO-X** 1.5, not something that can silently be presented as a recount for a later version.

Keeping the World You Found

After a successful generation, people wanted to keep it as well as admire it. **Q2PRO-X** 1.5 provided twelve ordinary global user slots and another twelve for the current mod. Six global and six mod-specific slots were reserved for music presets. Names could be edited in the menu, and updates were not supposed to overwrite these personal discoveries. Supplied packs, including HDR variants, offered a starting point for anyone not yet ready to build a scene from scratch.

A music preset saved the selected PROMODJ online-player track together with AVFX. A cold sky, rain or bright space scenery could thus have its own soundtrack. Loading an ordinary preset stopped the previous music preset's track: the new appearance did not inherit someone else's music by accident. The generator itself did not control music; it prepared a visual scene that could then be saved and arranged to taste.

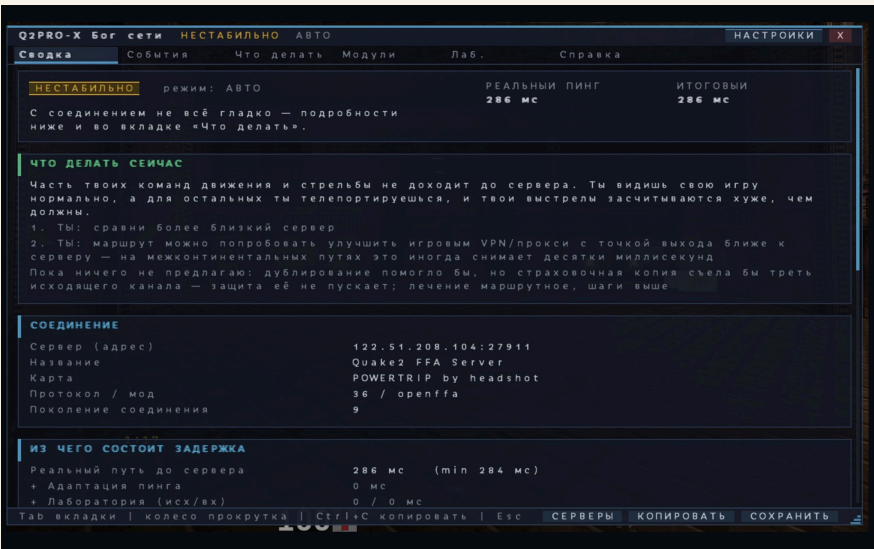
A preset carried another responsibility: loading it also changed saved drawings and volumetric steam on maps. A different world could not be loaded with half the previous one's smoke left behind. The complete set of placed authored content was therefore replaced, while the asset library and other slots remained the user's. A good preset became a finished scene worth returning to, rather than a fragile sequence of switches happened upon by chance.

The enormous number of possibilities did not imply that a person needed to see all of them. Finding one was enough: a familiar map worth lingering in, a reason to play the campaign again or a scene worth recording. The generator shortened the search. Saving a preset made the discovery last.

II. Developing the Client · Summer development and the 25 August 2026 release · Q2PRO-X 1.5

Between the Button Press and the Shot

A good ping does not guarantee smooth movement. One packet is delayed, the next catches up, and an opponent's positions arrive almost together even though the server sent them apart. A screenshot cannot show it; a crosshair makes it immediately apparent. **Network God** was created to investigate the connection. It collected measurements and helped choose settings, distinguishing what the client knew from what the received packets could not establish.



Network God presented network observations in a form players could use. Its operation was checked through repeated matches, including on unstable connections.

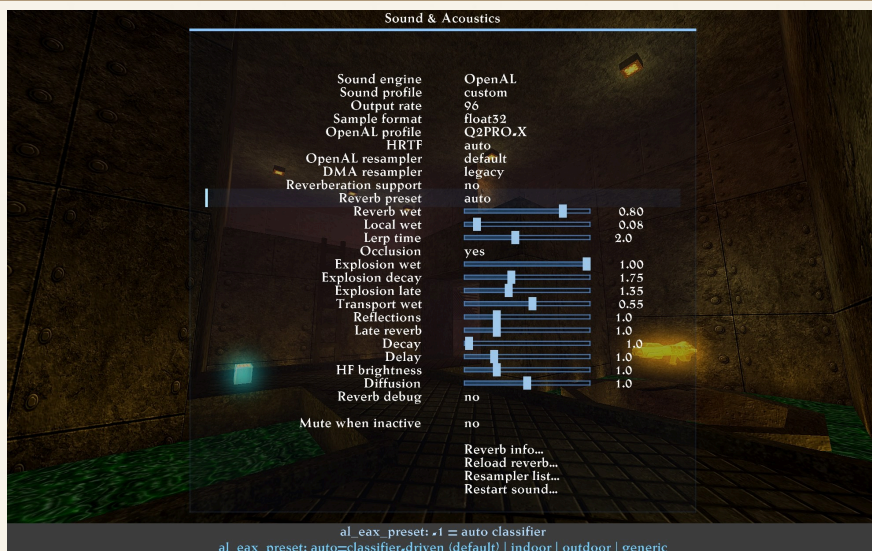
Eight live rounds of network testing took place in early August. An on-screen measurement could return to normal while an opponent still moved in jerks. Server-state frequency, losses, latency variation and client processing were examined separately. Automation was also found to keep changing behaviour after its corresponding setting was disabled. Both the automation and its explanations needed fixing: players had to understand what had been measured, what the program was waiting for and why it suggested the next step.

Network Lab compared connection conditions, Ping Adaptation allowed artificial delay to be tried, and laghax addressed visual response and corrections. Their readings were considered separately. The same caution would later be needed with bots: changing a scoreboard number is easy, but says nothing about when a bot actually saw an opponent or acted.

Network God combined several tools: finding a suitable client UDP port, Jitter Shield, Connection Guardian and observation of connection conditions. Uneven display of other players and artificially added latency were not rolled into one obscure number. Recommendations came with explanations, and a temporary change authorised by the user was tested with a way back. If nothing needed changing, the system was not supposed to invent work merely to keep its indicator busy.

Hearing the Match

Sound required equal care. **Quake II** players infer actions behind walls from footsteps, lifts and item pickups; a misplaced source or excessive echo changes their understanding of a situation. HRTF spatial processing and reverberation helped convey the surroundings, while separate levels controlled sound categories. Events still had to remain intelligible: an impressive explosion should not drown out everything useful. Voice chat added another independent stream. It needed testing after map changes, reconnects and transitions between activities, when devices and settings stayed the same but the game session changed.



Sound settings combined spatial processing with separate levels. In online play, distinguishing events mattered most.

The client gained online music and an intro with animation, a logo and its own tracks. The intro played once per program launch. It yielded to a direct server connection, and its music stopped when play began. Tests still found overlapping sound sources and incorrect image placement on wide screens—details heard and seen before a person had even started a match.

Music Between Matches

The online player could operate without music presets. PROMODJ offered genres, track searches, a queue, shuffle and an 'I'm feeling lucky' mode. The player and its window belonged to the client. Music could play in menus, pause for gameplay and resume on leaving a match; demo viewing had separate music settings. Ordinary **Quake II** tracks, the online player and intro music remained distinct sources.

Game audio was more than master volume too. Users could choose software DMA output or OpenAL, adjust spatial sound for headphones, room acoustics, audio profiles and individual weapon or player categories.

Resampling determined how a source recording was adapted to the output rate. The practical value was audible on a map: recognising a footstep and the direction of a shot mattered more than simply making everything louder.

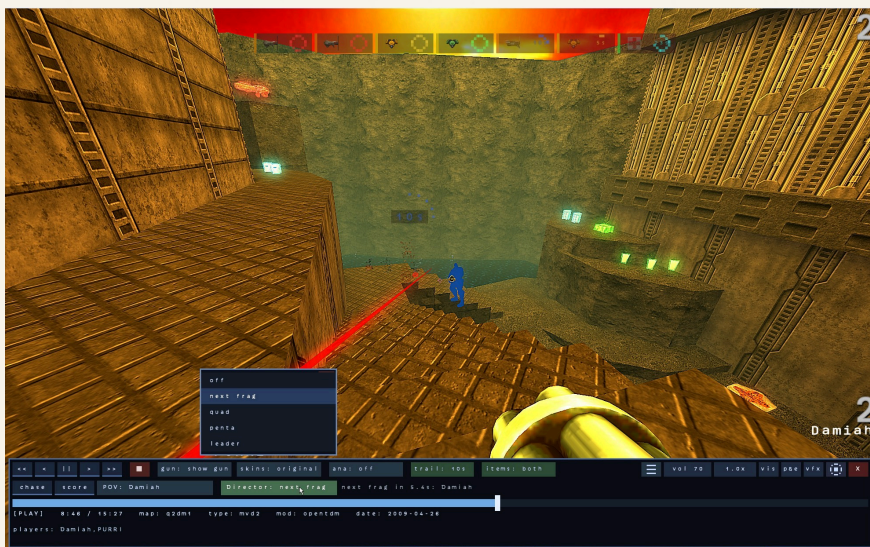
What the Intro Knows About the Player

The intro acquired a personal layer. Facts from the player's own matches appeared one after another beside the working menu: not generic praise for an unknown visitor, but information extracted from their recordings. Automatic statistics and fact display had to be enabled. If no suitable matches existed, the client invented nothing and did not fill the opening screen with zeros.

The logo had its own lighting, discharges ran around it, and a separate composition shaped the opening's duration and mood. Menu buttons remained available. The intro played once per launch; directly entering a map or server skipped it, and loading a game ended it. Online music autoplay could follow. The order mattered: two players should not compete for attention and volume at the same time.

Timers, Recordings and Statistics

An item timer can have several sources: a player's pickup, a manual countdown or spectator information. These were separated, with server permissions respected. A weapon dropped from a corpse is not a basis for timing a permanent spawn, while megahealth is linked to health decay and does not fit a simple formula. Changing the observed player required readings to be reconsidered. An incorrect bar could look convincing—and send someone to wait for an item at the wrong time.



Demo playback controls developed alongside the library and statistics. Recordings would later be used to investigate bots' tactical errors.

Returning to a failed encounter became easier: MVD gained next-frag navigation, camera controls and movement through the recording. Personal statistics supplemented the score, and the library helped find matches. Console messages were divided into categories with filters. A server address, error explanation or needed moment in a demo could now be found without reading the entire accumulated stream.

From a Demo Folder to a Match History

One recording does not always equal one match. It may contain warmup, spectating and actual play; a lack of frags does not prove that someone was watching. Personal statistics were therefore built from distinguishable events and mod states. DM2 and MVD2 support provided different amounts of information. When a recording could not establish a result, the answer was 'unknown', rather than a convenient invented zero in a table.

Seven statistics tabs offered different ways back into past games: an overview, match selection, comparisons by map, mode and opponent, search and checks of source-data quality. Users could restrict the period, specify their own nicknames, open details and jump to a recording. Weapon, frag and death information could expose what memory tends to smooth over: a favourite railgun might contribute less than expected, while the same corridor might end in death too often.

Analysis ran locally. Automatic scanning remained off until chosen by the user, updates were started separately, and collected facts could be cleared without deleting demos, favourites or playlists. Personal intro facts drew only on confirmed matches in which the player had made a positive contribution. Somebody else's MVD was not supposed to award the client owner somebody else's victories.

The demo browser remained a practical tool: library search and sorting, map and participant information, favourites, playlists and access to playback. The player let people revisit events, change speed and choose available viewpoints. Recording visuals were separated from ordinary gameplay: an episode could be watched with effects without rebuilding the familiar competitive appearance.

The bridge to old 32-bit game DLLs brought a separate puzzle. In the 64-bit build, a map loaded and the character moved, but megahealth behaved incorrectly. Changes to the item were not transferred in the same way as changes to the player. Synchronisation of affected entities was corrected in July. Reproducing the problem required collecting megahealth and following its entire cycle; simply loading the map did not reveal it.

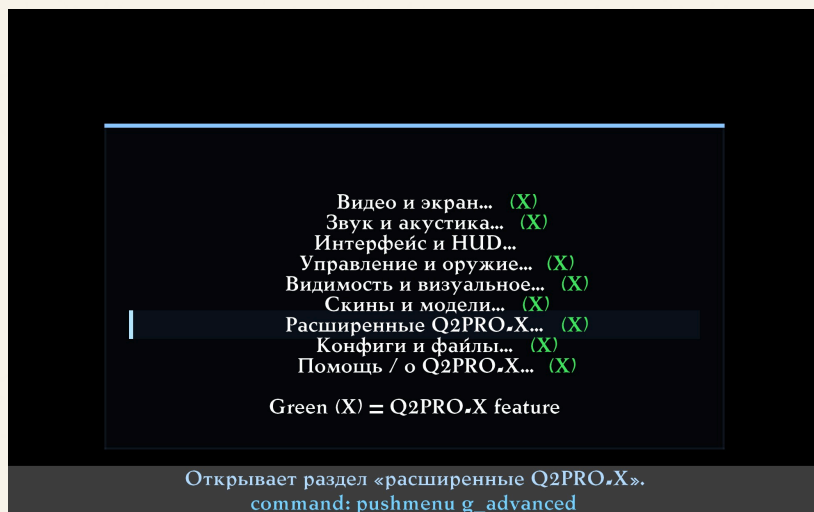
Hundreds of variables were already difficult to remember. Guided menus led to the right system with explanations, while the CVAR browser retained detailed search. Presets saved coherent visual arrangements, and the generator helped explore combinations. Factory values were separated from user choices, and settings for

particular areas from the full configuration. Otherwise, a launch or update might give people a different game from the one they had spent so long adjusting.

Settings You Do Not Have to Remember

The same concern for habit applied to the mouse and zoom. Movement-processing mode, zoom sensitivity, transition speed, a separate crosshair and walking while zoomed were independent choices. Importing an old configuration had to preserve the familiar feel of the hand. Bindings, Russian input, automatic transliteration and outgoing-message translation addressed less spectacular but everyday needs, from a short command to a conversation on an international server.

A modern monitor required choices too. True HDR, ordinary SDR, bloom, tone mapping and render scale were not one effect. OLED protection worked during inactivity; fullscreen, windowed operation and quick access to the desktop adapted the old game to a new workstation. Mods retained their own profiles, local visual overrides and command hints. Settings needed to survive transitions between these activities without an unexpected reset.



Q2PRO-X menus: a route to settings without having to remember their console names.

II. Developing the Client · Late August–10 October 2026 · Q2PRO-X 1.6 in development

Traces of Battle and the Cost of a Beautiful Frame

After **Q2PRO-X** 1.5, attention turned to impact marks. A stain on a wall stayed where the shot had hit; a wound on a moving model travelled with it. Player Contact Effects brought together blood, spray, the effects of sustained fire, bodies after death and a separate treatment for BFG impacts. The server still determined damage. The client assembled the visible battle from the events it received.

One hit could leave two stains. The client displayed a predicted effect, then received confirmation over the network and treated it as a new attack. Merging events too broadly caused the opposite problem: a real subsequent hit disappeared. The system began retaining the origin of evidence and matching confirmations. Entity numbers also needed care: an old wound could not be transferred to a new player who inherited a freed number.

A stain on a floor and a wound on a character need different coordinates. The first belongs to a level surface; the second must be attached to animated geometry. MD2 stores sequences of vertex frames, while MD5 uses a skeleton and weights. An approximate shell suitable for one format could slide or flicker on the other. Highlighting and control of model parts developed alongside this work: colouring a head, torso or another region had to match the actual pose, not an assumed shape. Testing covered moving players, their weapons, shadows and reflections.



Highlighting and model parts. A separate colour setting depended on geometry, animation, model-format support and several rendering paths.

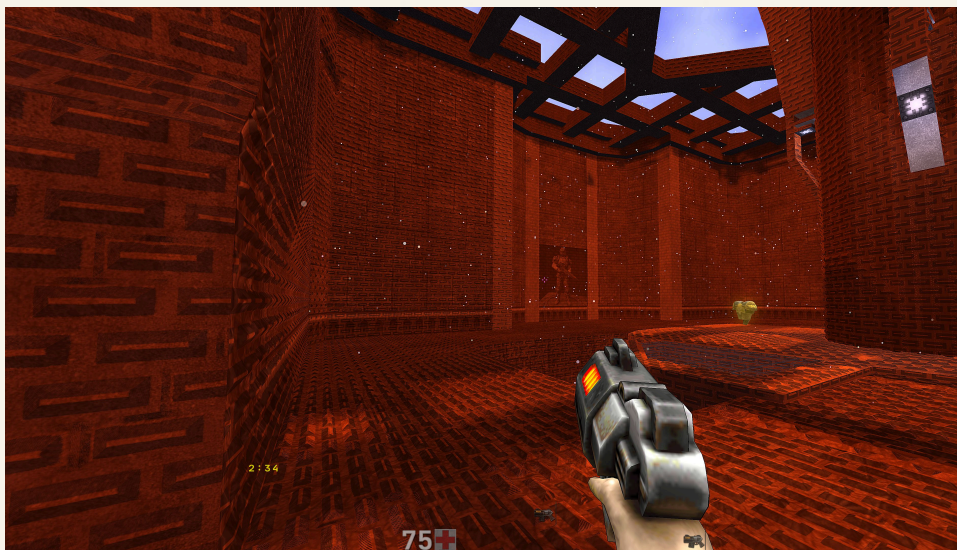
Player and item highlighting had existed since spring. Now it passed through more graphics layers: translucent colour could unexpectedly become solid; texture remained in one mode and disappeared in another. A screenshot could expose incorrect blending. Flicker was harder: sequences of frames had to be recorded and compared against model pose, depth and drawing order.

Blood on walls, water and AVFX puddles had different destination surfaces. Choosing the wrong floor left a stain in mid-air; an incorrect pool boundary spread it onto dry stone. Marks were checked on moving elements and in rooms temporarily out of view. Separate presets and settings slots let contact effects change while preserving the rest of a scene. The player's own blood was adjusted so it did not obscure the fight.

Stalls When the Camera Turned

By autumn, a high FPS figure no longer adequately described the client's speed. Brick geometry could render quickly for a long time, then unexpectedly rebuild as the camera turned. The cache discarded recently

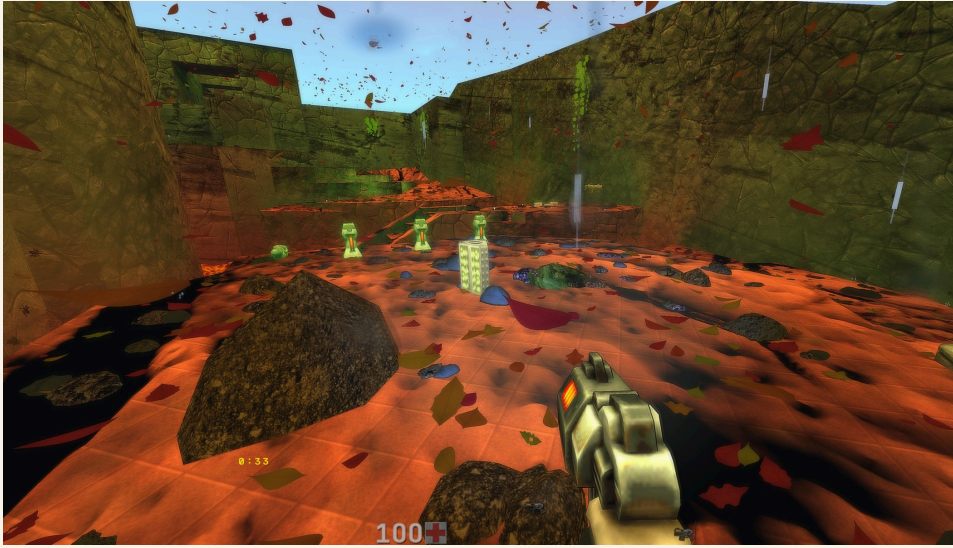
used data; looking back forced it to be built again. A diagnostic run counted 329 builds and 320 evictions, with individual delays approaching half a second. Making one operation faster would not solve the main problem: expensive preparation kept returning inside gameplay frames.



Volumetric brickwork in Q2PRO-X 1.6: projecting bricks and recessed joints give the wall depth. Preparing this geometry became one of the tasks of the autumn optimisation work.

Stable data began to be prepared at map load or when a setting changed. Camera turns no longer continually rebuilt bricks, but the cache occupied about 1.7 GB on one demanding test map and 3.3 GB on another. Initial preparation had not disappeared either. The gain was in removing it from ordinary movement; smoother frames were paid for with memory and waiting before play.

Reflections exposed another source of unnecessary work. The world was rendered again for several planes, with complex skies calculated before walls that subsequently covered them. Drawing the sky moved behind opaque geometry, using the depth already available. In the heaviest measured case, the improvement was only from 43.6 to 44.6 frames per second. There was less wasted computation, but the reflected world remained expensive. Other effect combinations continued to be tested separately.



Autumn scenery served both as a distinctive look and as a demanding measurement case. Weather was allowed to reach the same state before comparisons.

An autumn preset helped test the rejection of invisible work. Skipping what the camera cannot see is useful; stopping puddles from developing there is a bug. Rain must keep filling them without an observer. Separating world state from its presentation raised the average frame rate in one controlled, demanding scenario from 75.5 to 86 FPS, roughly fourteen per cent. Comparisons included the same wait in rain, tens of thousands of variants and frame checks. Runs that lost focus or activated the screensaver were excluded: they were not measuring the intended comparison.

The current **Q2PRO-X** 1.6 line continued work on particles, lighting, reflections, vegetation and seasonal materials. Performance HUD showed their cost, while render scale and post-processing helped tune the load. On 10 October, the version was still developing. Meanwhile, another kind of testing had appeared: instead of comparing frames, it required rewatching matches and asking why a new computer opponent had lost.

Effects, Individually

Q2PRO-X 1.6 divided ordinary particles into 68 Standard families. Blood, blaster trails, rocket smoke, sparks, teleports and railgun particles could be enabled, disabled and coloured independently. Ten ordinary AVFX families gained settings of their own: changing a standard weapon trail should not quietly recolour the atmosphere. Sky meteors, falling meteors and comets retained separate colour choices—they are different phenomena, even if each leaves a luminous trail.

Lighting and object visibility gained their own presets. Gameplay choices included bright, saturated, monochrome and ‘Nothing Extra’ looks, with separate lighting choices for AVFX. Such a preset did not enable rain or plants on the user's behalf; it set lighting conditions. Model highlighting, individual parts and ordinary bloom were saved in their own set. The atmosphere could change without losing the carefully chosen readability of an opponent.

Optional health-and-armour music ambience also appeared. Nine states tied the background music's mood to the player's protection, with crossfaded transitions. Single-player and deathmatch were enabled separately, and an explicitly selected online track or music preset took precedence. It offered another way to feel the character's condition, not another source of information about opponents.

Users could replace audio assets and intro music. Custom intro tracks required neither rebuilding the client nor using factory filenames: language-specific FLAC collections were supported, with duration taken from the file itself. If no suitable music existed, the visual intro remained. Packaging details belonged in the guide; what mattered to players was that their chosen appearance did not have to sound exactly as the release authors intended.

A small but useful addition retained the history of successful connections. Shared across mods, it let people return to the last server after restarting, or reconnect automatically if desired. Failed attempts and mere ping refreshes did not clutter it. Such a feature is easy to overlook beside complex graphics, though it shortens the journey from launching the client to playing every day.



Q2PRO-X contact effects connect lighting and particles with events in combat.

III. Opponents and New Tools · September 2026 · Early betas and OpenTDM-X Beta 10

OpenTDM-X: Teaching a Bot to Win

A bot could hardly miss and still lose the map. It surrendered red armour, emerged from the lower floor under an opponent's railgun, then respawned unarmed in a room where someone was already waiting. After a few such encounters, the human had everything needed and could choose the fights. **OpenTDM-X** set out to create an opponent capable of breaking that pattern: recovering, taking away a favourable position and exploiting someone else's mistake.

Earlier bot projects were studied in September. Candidates included JABot from the ACE family and 3ZB2: one offered appealing separation between subsystems, the other detailed movement and combat cases. Other projects had limitations too. The **Gladiator** archive lacked the source for its intelligence library, while **Eraser** was missing some navigation code. Even complete source did not provide a ready-made, physically fair **OpenTDM** player. Some bots sent ordinary commands while also changing velocity directly. Importing an entire controller like that would quietly grant different movement capabilities.

OpenTDM supplied the rules: weapons, items, physics and scoring. Perception, goal selection and execution were written for the bot. Its skill parameter determined capabilities, including permitted knowledge of opponents. Even at high levels, that knowledge did not grant different physics. A real explosion propelled a rocket jump, and a fired rocket travelled the same path as a human player's rocket.

What a Match Recording Can Teach

q2duel5 was the first training map. A September analysis pass decoded 121 unique recording streams—more than a million rows of player state. The program found demos both in growing folders and inside archives, without reanalysing streams already read. Decisions then had to be reconstructed from those frames: which item a player sought, why they changed direction, what they did before shooting and where they went afterwards.

A successful escape is not always a useful lesson. A player may survive because of the chosen route—or because the opponent missed. Fragments were compared with other recordings and level geometry. Tricks were broken down into acceleration, take-off and landing: the action had to be reproduced, rather than simply arriving at the recorded coordinates.

In early tests, a bot approached the railgun and jumped away, leaving it behind. It dodged one grenade into the path of the next. It collected a chaingun but forgot the ammunition beside it. The log could dutifully report selecting a goal; the human saw an uncollected weapon and picked it up. The investigation had to find what interrupted an action one step from success and keep a nearby worthwhile goal active until completion.

After spawning, reach a weapon



Leaving the line of fire preserves the next chance

A route after spawning: reach cover and a weapon while retaining the ability to answer the opponent's fire.

By **OpenTDM-X** Beta 10, the bot knew compact q2duel5 well. It applied pressure, caught respawns and fired into exits. That build was kept for comparison: if new rules seemed to slow the bot down, it was run alongside the newer one and played again. Some complaints concerned older weaknesses. In public Beta 10 demos, a human could also strip its stack from above while it remained on the lower floor. Returning to the older build would not solve that.

With full control, a human began hunting spawn points. The bare bot again answered with a blaster from a poor position and died, although it could have moved towards weapons and armour while firing on the way. One proposal introduced an increasing pause after repeated quick deaths. In a match, it helped the opponent, who could collect the whole level before the next spawn. The rule was removed and a one-second respawn restored. The mistake became clear across a series of deaths; in isolation, the delay did indeed postpone one death.

III. Opponents and New Tools · Late September–4 October 2026 · OpenTDM-X Beta 11

Weapons and Four Personalities

A human advanced at point-blank range with a hyperblaster while the bot held a rocket launcher and stayed silent. It had not frozen: each rule was faithfully doing its job. One forbade wasting time switching weapons; another forbade firing at the risk of killing itself with the blast. Together, they produced the one outcome the player had insisted must never happen: several seconds under fire without a reply. The investigation worked backwards from a ready shot—what could the bot do now, and which decision deprived it of that opportunity?

Switching could be fatal too. A rocket almost finished a human off; the bot drew a super shotgun and died before the change completed. The time to the next shot from the raised weapon had to be compared with the time to the first shot from another. The railgun remained useful for an opening hit and catching a respawn. But in a close pursuit of a wounded opponent, a burst or a spread often hit more reliably. One list of ‘best weapons’ could not cover all these situations.

Chasing Around a Loop

On a looped map, a human ahead on score could stop fighting altogether. They listened for the bot and left by another corridor. The chase repeated, circuit after circuit. Interception needed route changes, exit control and quiet waiting; even the railgun mattered, because a held weapon can give itself away by sound.

Once a weak opponent appeared in direct view, however, a detour lost its purpose. An elaborate manoeuvre could give the human time to rebuild their stack.

A similar choice followed a frag. An available shot at a respawn on compact q2duel5 was worth taking immediately; chasing across a larger level left armour and weapons unclaimed. Small items mattered too. A stimpack or shard could raise a stack beyond the threshold of a single railgun kill, giving an unarmed opponent another chance to escape. Item value was tied to travel distance and the next likely encounter.

Four Styles

Four styles appeared: default, arcade, defence and chaotic. The first retained the main tactics; the second favoured fast movement and fighting; the third favoured survival, stack and control. The fourth varied its preferences during a game. A style could be set in a bot's profile and changed temporarily by vote. Their introduction required another comparison of versions: default had grown weaker, while defence sometimes refused even an available frag.

Arcade initially exposed an overly literal reading of attack: a bare bot ran at a prepared human. Active combat still requires a weapon, ammunition and protection; useful items along the way had to retain their place in the route. Defence made the opposite mistake, releasing an almost-finished opponent and ignoring a safe pickup. Changes to both styles returned to q2duel5, where speed and pressure already had a standard for comparison.

The shooting scale below skill 100 was checked separately. Low levels sometimes retained such accurate railgun aim that difficulty differences lost meaning. Accuracy reduction was made gradual without altering shooting at 100 or above. Adaptive ping had to be traced from behaviour to the server's response to an external browser. The mod's scoreboard could show latency for a bot while the browser still showed zero. Correcting that number was checked separately from perception and action timing.

OpenTDM-X Beta 11 was accepted for public installation on 4 October. Styles, weapon behaviour, pressure, escapes from poor positions and new profiles formed one package. Its change list compared it with the server's Beta 10. Long chains of internal fixes were reduced to their outcome: a server operator needed to install an update, not reconstruct its history from intermediate archives.

Four personalities for one opponent



Default

Combat and item control



Arcade

Attack with a weapon



Defence

Supplies, cover and survival



Chaotic

Changing familiar decisions

Play styles change decision priorities while sharing the same shooting and movement rules.

III. Opponents and New Tools · 4–10 October 2026 · OpenTDM-X Beta 11.1 · OpenTDM-X Beta 11.2

Red Armour, Megahealth and Teleports

On ztn2dm3, the bot that played q2duel5 confidently often looked like a beginner. Here there was red armour, megahealth and open space. A rocket had travelled only part of the distance while the human had already fired a rail and stepped aside. The bot collected a weapon but left the megahealth, missed an item jump and walked away, or lost seconds on the stairs by adrenaline. On this map, its former accuracy could not compensate for the stack it surrendered.

One human route took only seconds: rocket launcher and ammunition, yellow armour, then a rocket jump to the chaingun. Next came its ammunition, two health packs along the way and red armour. Health partly repaid the jump's cost, and two strong weapons were now available. The bot was taught to complete this sequence quickly and adjust it if an item had already been taken. Early versions instead wavered between goals; during that hesitation, the human reached the red armour.

At the step before the railgun, the bot learned the trick and still sometimes left the weapon behind. It landed on the platform, noticed another goal and jumped away; elsewhere, it repeated the jump despite the railgun having just been collected. The whole sequence needed checking: starting position, acceleration, item contact and the next step. A graceful landing alone no longer counted as success.

Lifts and Fire into Cover

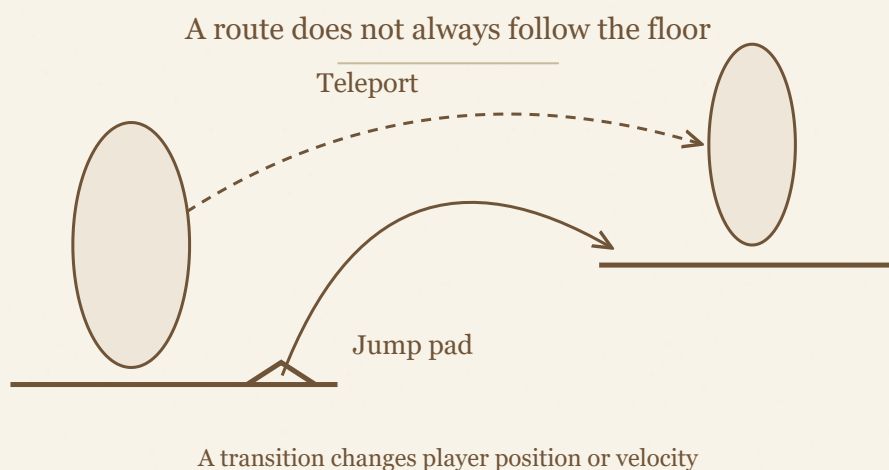
Lifts exposed several independent weaknesses. Waiting-for-platform state prevented the bot from finding a firing position even while an opponent rose directly in front of it. A rocket could fly towards where the human stood when fire was pressed and strike the floor after they had ascended. Platform speed and direction, flight time and possible movement changes had to be considered. If a player's protruding body hit an obstacle and the lift stopped or reversed, the old prediction was no longer valid. Falling under a platform required separate navigation recovery; better aiming could not solve it.

Rockets and grenades also had to be checked against room geometry. A human in the megahealth niche could hide their model while remaining within blast range of a wall. A grenade launcher could bounce a projectile around a corner; aiming directly at the body discarded that advantage. Yet endlessly firing into an empty exit consumed every round. Recordings showed both faults: silence beside an occupied niche and shooting the last rocket at somewhere the opponent had long since left.

Individual profiles developed in **OpenTDM-X** Beta 11.1 as well. Chat lines came from available references, greetings accounted for the current person and connection history, and replies to smileys ceased to be a constant echo. One mixed profile gained special behaviour when far ahead: do not immediately attack an unarmed respawn, approach, drop a weapon with ammunition and allow a few seconds to settle in. This rule applied only to that profile. A projectile already in flight did not become harmless—friendliness did not rewrite physics. Other bots continued the ordinary contest for the result.

An Ubuntu 22.04 build was prepared for public **OpenTDM-X** Beta 11.1. Package checks found an outdated example server configuration and unrelated client files in the source archive. The packages were rebuilt, separating the mod from the rest of the working folder. Installation and documentation were checked on both platforms.

OpenTDM-X Beta 11.2 with q3t2 was accepted and released on 8 October. Teleports moved players instantly and jump pads sharply changed velocity; those transitions could no longer be judged as ordinary running or a broken route. Afterwards, the bot continued towards its next goal. Previous maps remained in testing. On ztn2dm3, moving rocket jumps still needed speeding up, long-range combat choices needed work and delays around adrenaline remained. The next, still unfinished **OpenTDM-X** Beta 11.3 was addressing them.

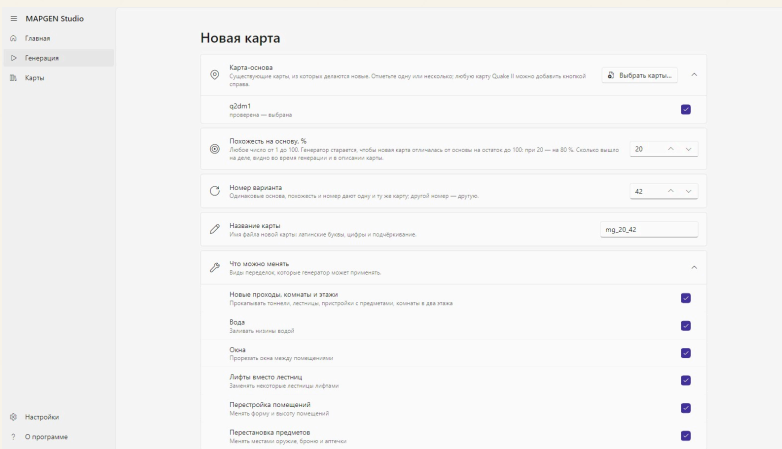


A teleport and a jump pad interrupt an ordinary route in different ways. This diagram explains the transitions, not q3t2's geometry.

III. Opponents and New Tools · Autumn 2026

From Studying Demos to Building Tools

Analysing everything by hand was becoming harder. **Mapgen Studio** emerged as a separate application for maps: base layouts, donors, transitions, a library and checks. **Bot Training Studio** took on demos and geometry, analysis and training. Users would add recordings and obtain knowledge files. They would not need to rebuild the game library for every new map.



Mapgen Studio is a separate application for working with maps. Its development alongside the client helped shape the requirements for the next studio.

The model in **Bot Training Studio** works before a match. It can spend time analysing material on a GPU; the finished server bot must respond without that pause. Training was separated from execution. Model sizes were matched to available memory, with NVIDIA, AMD, Intel and a CPU fallback investigated. Readiness differed between options: seeing a device in a menu did not mean training, performance and results had been verified on it.

The studio's first launches brought down-to-earth problems. While Python was being checked, controls appeared and disappeared in the main window, and a single graphics card appeared several times in the list. Environment checking moved into the startup process. Libraries were included in the package, with installation of a compatible Python available when needed. Russian and English interfaces, hints, version history and updating followed.

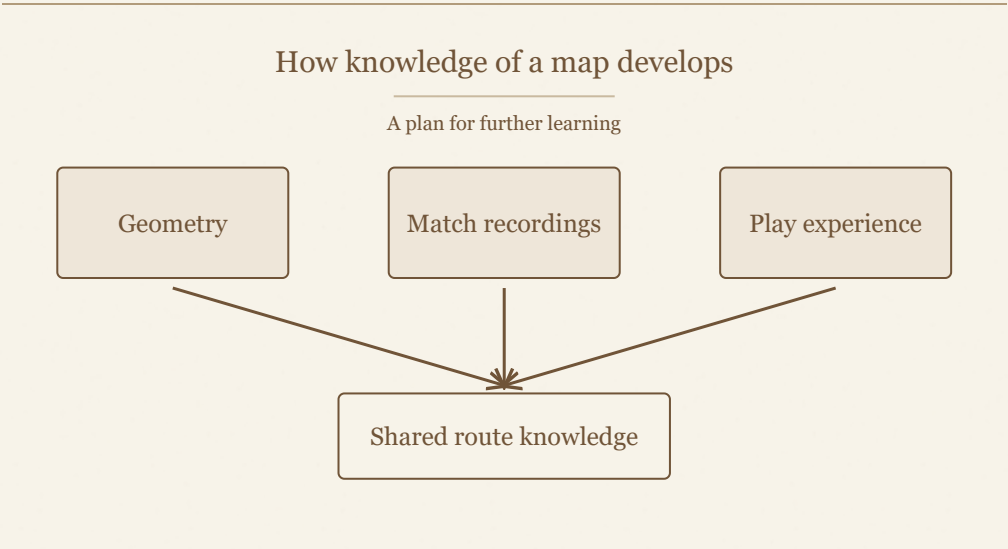
Long-running tasks moved into the background: the window remained usable, jobs could be cancelled and exiting produced a warning. Indicators showed CPU, GPU and memory load. The factory model was separated from user additions; otherwise, an application update would overwrite hours of somebody else's training. Export began to account for data provenance and compatibility, while interrupted jobs retained their saved state.

A Teleport on a Graph

After q3t2 was accepted, its matches and separate trick recordings joined the training programme. A large coordinate jump on a graph might be a teleport, respawn, jump pad or recording gap. Mixing them would teach the model the wrong movement lesson. Events were therefore matched against trigger geometry, exit directions and subsequent states. The verified material included 64 full match recordings and 29 movement recordings. They showed not just where a person moved, but which mechanic made that movement possible.

Models were trained on NVIDIA and checked against held-out recordings. Some movement and transition measures improved. Related demos stayed on the same side of the dataset split: neighbouring frames from one match are too similar to count as an independent examination. Weapon choice, resource control and trick execution remained unresolved. On 10 October, the studio could already run training, but its results could not yet automatically produce a complete, strong duellist.

A baseline for unfamiliar maps was planned next. A bot would study routes and items during warmup, save a foundation shared by all profiles, then enrich it through matches. The studio would accept those files or start from geometry itself, with or without demos. A separate competition design proposed fixed conditions and comparisons between versions. At the book's cutoff, both remained plans.



The plan for further learning combines geometry, demos and match experience. This diagram describes an intention, not a finished feature.

The First Launch

After installing **Q2PRO-X**, you can begin with the same configuration and server you used before. New settings do not demand an immediate change to the way you play. Then, on a free evening, open a familiar map without an opponent and walk through it more slowly: watch rain drain from an upper platform, see the Moon reflected, or examine a passage under different light. Or open one of your demos and finally discover how an opponent managed to arrive first.

A match usually calls for readable models, distinct sound and a steady response. During a walk, there is room for a different sky, weather and wildlife. Bots offer more things to try: repeat a trick, practise defence or arrange an open duel. Work on them continues, including through recordings in which a human has found yet another way to outwit them.

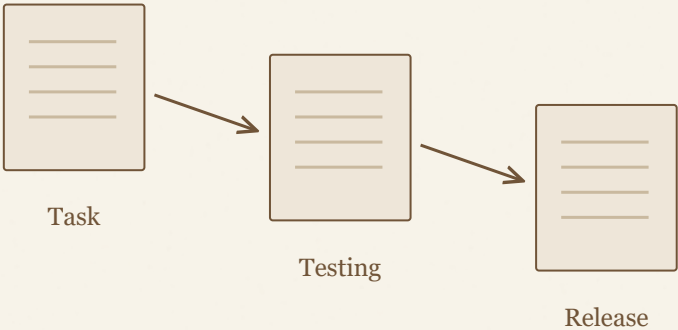
Builds and guides are available at q2pro-x.com. A good place to begin is a level you know by heart: walk to a weapon, take a lift, check movement and fire a shot. Changes are easier to notice there. Then you can join a server. Once somebody is collecting armour behind the wall, there is no time left to browse the menus.

Appendices · As of 10 October 2026

Counting the Work Behind the Project

Q2PRO-X's first records date to 29 March, and this chronicle ends on 10 October 2026. Counting both dates gives 196 calendar days. In that time, the client grew from early checks of controls and weapon prediction into a broad collection of graphics, networking and user features, alongside a bot mod and independent studios. The main dates below provide bearings in that crowded history. Client version numbers and **OpenTDM-X** beta numbers belong to different development lines and should not be read as a single sequence.

From the first assignment to release



Assignments and repeated checks lie behind a release. The numbers in this chapter describe different parts of that process.

Period	Developments
29 March	Q2PRO-X begins. Early versions, weapon response, controls and compatibility.
April	Interface, demo viewing and communication develop; public Q2PRO-X 1.2 on 29 April.
Late April–May	Intensive AVFX, rain and snow development; public Q2PRO-X 1.3 on 17 May.
May–early July	Sky, water, HDR and local modes; preparation of Q2PRO-X 1.4.
July–August	Materials, wildlife, Network God, timers, compatibility and extensive Q2PRO-X 1.5 testing.
25 August	Q2PRO-X 1.5 published; the next client line, Q2PRO-X 1.6, opens.
September	Surfaces and scenery expand, demanding scenes are optimised and duel-bot work develops.
4–6 October	Public OpenTDM-X Beta 11 and Beta 11.1; further work on ztn2dm3 and styles.
8 October	OpenTDM-X Beta 11.2 with q3t2 accepted and released; the map joins Studio research.
9–10 October	Q2PRO-X 1.6, learning research and ztn2dm3 improvements in OpenTDM-X Beta 11.3 continue.

What the Archive Can Tell Us

196

days of development in this chronicle

776

task documents

4 739

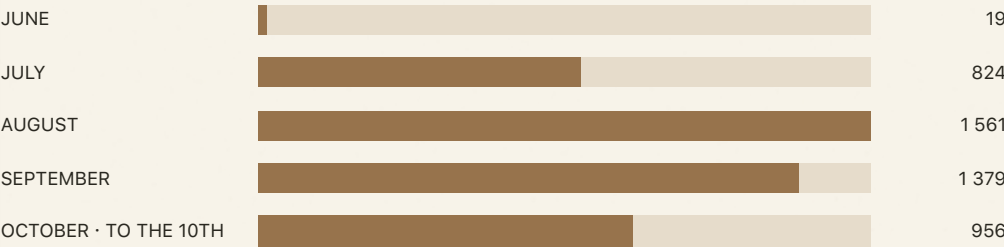
commits touching code

99

groups of project capabilities

The examined archive contained 776 documents setting out tasks. Of these, 520 were main or follow-up assignments, and 256 held accompanying requirements and corrections. Exact content duplicates were excluded; reports of results were not counted as new assignments. This is a document count, not a count of independent features. One capability could generate a long chain of clarifications, especially when a fix returned from an unsuccessful test. The examined documentary collection contained 3,245 Markdown files in total, of which 3,110 had distinct contents.

A snapshot of four related Git trees yielded 9,294 unique commits, 4,739 of which touched code within the chosen scope. Initial source imports, third-party files and automatically generated files were excluded. Available branches included experiments and intermediate builds. A commit here therefore means a recorded change, not a separately finished feature. One idea might pass through several rewrites; a change transferred under another hash also remains a separate record.



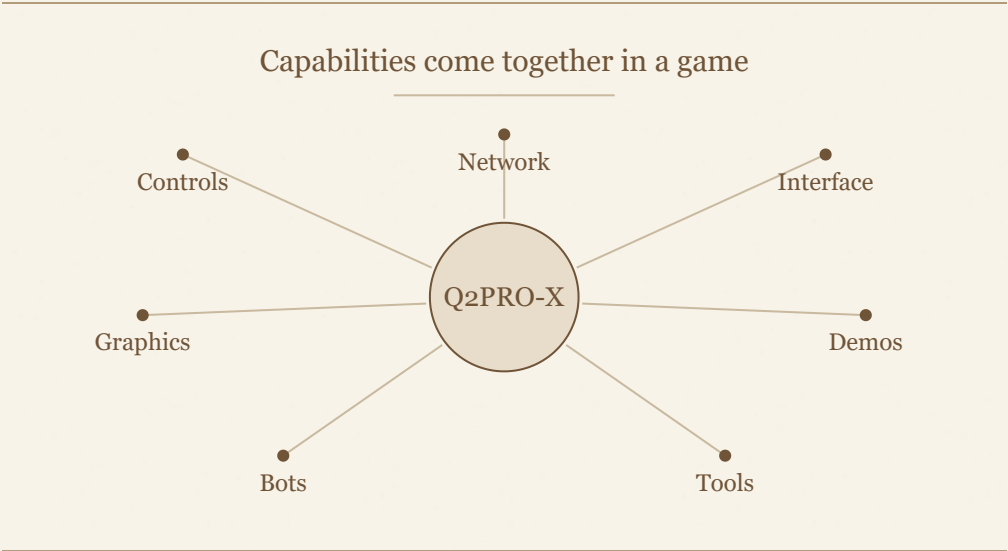
The chart begins in June because the available code history does not cover March–May in a comparable form. An early tree import cannot be presented as the entire project having been written at once. October ends at the book's cutoff. The same snapshot counted 984,895 added lines, 102,743 deletions and 2,735 code-file paths. These describe editing events: a line could be rewritten repeatedly or removed after an experiment. They are neither the final program's size nor a measure from which development productivity can be calculated.

Time Spent Testing Again

The archive preserves many checks in detail, but it is not a continuous timesheet. It cannot honestly yield the total hours of development, repeated matches, demo analysis and waiting for builds. A report date does not say how many uninterrupted hours the work took; a demo's duration is not the time spent studying it. Instead of an invented enormous total, this book offers concrete evidence: successive early betas, repeated failures after material fixes, eight rounds of network tests, demanding-scene comparisons and dozens of public recordings used to investigate tactics.

Some work inevitably had to be repeated: revisit an episode already fixed, retake measurements with matching configurations, launch an older build for comparison or play another match after changing one rule. Five returns to material re-enabling left users with one working setting. Faster shader preparation added no menu item, but shortened the wait. The change history reflects that work; counting only new features scarcely notices it.

The project's capabilities could be grouped into 99 subject areas. Successive fixes to one function were not counted as new features, and large effect families were not divided into every slider. The list includes everyday player tools as well as independent applications for preparing maps and training bots.



The next section groups capabilities by their use, from controls to preparing maps and bots.

Appendices · The client, the mod and independent applications

What You Can Do in the Game

There is no need to enable everything on a first visit. Begin with familiar controls and a server search, then watch one of your match recordings or change the weather on a map you know. The following pages gather the capabilities discussed in the book according to what they let a player do.

Adjust Movement and Shooting

A player presses fire and wants to see the shot immediately. **Q2PRO-X** can display it before the server's reply arrives: this is weapon prediction. The server still decides whether the projectile hit and how much damage it caused. Another system, Laghax, softens jolts when the server corrects a player's position. Its indicator shows latency, its variation and prediction state.

Mouse handling and a supported movement mode can be selected, a familiar fullscreen view retained or windowed play used. Players accustomed to 4:3 need not stretch the picture over a wide monitor: bars remain at the sides. Render scale and subsequent processing offer a choice between sharpness and GPU load.

Find a Server and Talk to Your Team

The server list supports searching for a game, filtering unsuitable choices and saving favourite addresses. In a match, voice chat removes the need to type while running. Microphones are selected in the game, and speech can be

transmitted by holding a button or by voice activation. **OpenTDM** team matches gain additional teammate information on screen. Connection history makes it possible to return to a visited server after restarting the client.

When movement suddenly becomes uneven, it helps to distinguish a slow computer from an unstable connection. **Network God** settings, ping-adapted prediction and **Network Lab** tools allow those cases to be examined. Average latency is not the only important measure: how much it changes between packets matters too.

Understand Settings and Keep Your Habits

There is no need to remember hundreds of console commands. Menus with explanations lead to the required setting; a separate game-variable search reaches the details. The interface can switch between Russian and English. Fonts, startup presentation and personal frag messages also form part of client customisation. Console messages are categorised to make useful lines easier to find among service output.

An old configuration should not obstruct an update. The client therefore retained compatible ways of handling settings and separated preferences for ordinary play, mods and demo viewing. OLED screens have protection during inactivity.

Players and items can be highlighted, including individual model parts. Timers help track item returns where the game mode permits them. When an attractive scene becomes too demanding, the performance display helps identify where frame time is being spent.

Change a Familiar Map's Weather and Appearance

Rain and snow can fall on a familiar level. Water collects and reflects the surroundings, snow changes the surface and drops appear on a wet lens. Lava, foam and bubbles have their own effects. Materials, vegetation, ivy and autumn looks transform the same walls, courtyards and stairs in different ways.

Above the map, there can be weather skies, stars from an astronomical catalogue, the Moon and other celestial bodies. Meteors, comets, eclipses, black holes and sunbeams are available. High-dynamic-range, lighting and visibility settings determine how they look beside a dark passage or bright explosion.

Impacts, blood and other traces of combat connect the image with events in a match. Particle groups are adjusted separately. Every combination need not be assembled by hand: supplied presets and the generator let people choose an overall look and then refine it. Surfaces can gain hardware relief, mirrors and personal drawings; the air can gain volumetric clouds, fog and placed steam. Hardware tessellation depends on the selected graphics backend.

Revisit a Match or Play Without a Server

A recording can be found in the library, opened in the expanded player and navigated to an interesting moment. Multiview recordings support jumping to the next frag. Personal statistics offer another view of past games, while playback visuals let a scene be examined differently from the way it appeared during combat.

Sound adapts to the way people play: spatial processing for headphones, room reverberation and separate category volumes are available. Online music exists alongside game sounds without

requiring their master volume to change. Music presets pair a scene with a track, while optional ambience changes with the character's health and armour.

A dedicated game DLL can alter single-player: shuffle items and monsters, turn water into real lava and reverse the change. Monster appearance and visible geometry can vary separately. Multiplayer maps support local monster training with item respawns and scoring; **Free World** offers more room for walking and observation. A bridge to 32-bit game modules helps run older additions.

Choose a Computer Opponent

OpenTDM-X is a separate mod with duel bots. Opponents have a skill level and style: balanced, offensive, defensive or changeable. A style can be attached to a profile and changed temporarily before a match. Bots learn to gather weapons and armour, finish a favourable fight, escape exposed positions and use suppressive fire, tricks and rocket jumps.

On q2duel5, narrow passages and pressure after respawning are especially important. ztn2dm3 adds red armour, megahealth and dangerous open areas. q3t2 requires teleports and jump pads to be understood. Yet choosing weapons, collecting ammunition and seeking cover must work on any map.

Lower difficulties receive less accurate shooting; ping adaptation accounts for delay when playing a human. Profiles have their own chat lines: greetings, occasional smiley replies, introductions and help. One mixed personality can ease pressure when far ahead and share a weapon with an unarmed opponent. Mod packages are available for Windows and Ubuntu 22.04.

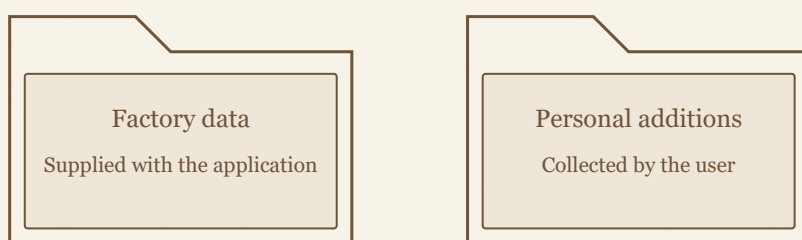
Prepare a Map and Training Material

Mapgen Studio runs separately from the game. Users select a base map, add parts of other levels, connect transitions and check the result. Its library helps store prepared variants and return to them.

Bot Training Studio has a different purpose: loading match and trick recordings, analysing movement and preparing material for local training. Factory data and user additions are kept separate so an update does not erase personal work. Long jobs run in the background; the interface remains available and shows resource use. GPU and CPU options, version history, updating and a prepared runtime environment are provided.

By the date of this book, demo analysis and initial training experiments were working. Producing complete tactical behaviour for a new map still required research and in-game checks. The studio is not presented as a finished, one-click way to create an unbeatable opponent.

Updating must not erase learning



Two independent sets of files

Keeping factory data and personal additions separate protects the user's work when the studio is updated.
