# cecris

Public

Notifications  Fork 0  Star 0

main  Go to file  Code

**zorel** Add graceful SIGTERM maintenance mo... 8b56fe4 · 3 days ago

| | | |
|---|---|---|
| src | Add graceful SIGTERM... | 3 days ago |
| tests | Add graceful SIGTERM... | 3 days ago |
| web | Add graceful SIGTERM... | 3 days ago |
| .gitignore | cecris 0.1.0: real-time ... | 3 days ago |
| Cargo.lock | cecris 0.1.0: real-time ... | 3 days ago |
| Cargo.toml | Add Apache-2.0 license | 3 days ago |
| LICENSE | Add Apache-2.0 license | 3 days ago |
| Makefile | Add graceful SIGTERM... | 3 days ago |
| README.md | Add graceful SIGTERM... | 3 days ago |

README  Apache-2.0 license

# cecris — web edition

Server-authoritative, real-time multiplayer falling-block battle: a Rust server with zero external crates and a browser client, talking a compact binary protocol over WebSocket with an SSE + POST fallback.

Rooms are addressed by game id, so a single link ( `/g/<id>` ) is all a group needs. Any number of rooms run in parallel on the same process.

## Quick start

```
cargo run --release -- --bind 0.0.0.0 --port 808
# server: http://0.0.0.0:8080
# client: http://localhost:8080
```

Listen address and port are command-line options; environment variables provide the defaults when a flag is omitted:

| option | env | default | meaning |
|---|---|---|---|
| `-b` , `--bind` `<ADDR>` | `BIND` | `0.0.0.0` | listen address |
| `-p` , `--port` `<PORT>` | `PORT` | `8080` | TCP listen port |

```
./target/release/cecris --bind 127.0.0.1 --port
./target/release/cecris --port=9001          # -
./target/release/cecris --help
```

Open the site, pick a nickname and a game code, then share the room link. The first player in a room is the host, controls the round options, and can start once at least two players are connected and everyone has pressed **Ready**. Two to six players per room; late joiners enter the next round.

## Controls

| key | action |
|---|---|
| ← / → (A / D) | move, with DAS/auto-repeat |
| ↑ or X | rotate clockwise |
| Z | rotate counter-clockwise |
| ↓ (S) | soft drop |
| Space | hard drop |
| 1–5 | select a special from inventory |
| click a field | use the selected special on that field (own included) |
| Esc | cancel special selection |
| Enter (chat box) | send chat |

The own board is client-predicted for horizontal moves and rotations; the server reconciles at 20+ Hz. Opponent boards and all rule changes are server-authoritative.

## Rules

A 12 × 22 field with seven piece shapes and classic rotation tables. Pieces are dealt from a per-player 7-bag: all seven shapes are shuffled, dealt once each, then the bag is reshuffled. This limits droughts and long runs of the same shape compared with picking each piece independently.

A **special block** is a lettered block embedded in a stack. Clearing a row that contains one adds it to your **inventory**, the list beside your board. Select an inventory entry and click a field to spend it there; useful specials can be used on your own field. The inventory starts empty and holds only collected specials.

The host edits the room's options in the lobby; every joiner sees the same values read-only and the settings carry over between rounds. Editing locks when the countdown starts. The options panel is collapsed by default; guests only see it when at least one option differs from the defaults.

Every connected player must press **Ready**. The host can start only when everyone, including the host, is ready. Starting runs a 3-second countdown; joining or leaving in a way that breaks the ready set cancels it back to the lobby. Readiness resets after each round. The lobby also shows a short how-to-play card.

## About

Real-time multiplayer falling-block battle: zero-dependency Rust server with WebSocket + SSE browser client.

game  multiplayer  rust

sse  websocket

Readme

Apache-2.0 license

Activity

**0** stars

**0** watching

**0** forks

Report repository

## Releases 1

**cecris 0.1.0** Latest

3 days ago

## Contributors 1

**zorel** Aurélien DEHAY

## Languages

**Rust** 54.2%  **JavaScript** 37.6%

**CSS** 7.1%  **Other** 1.1%

| option | default | range |
|---|---|---|
| Start level | 1 | 1–15 |
| Starting garbage rows | 0 | 0–12 |
| Lines per level | 10 | 1–50 |
| Level step per threshold | 1 | 1–5 |
| Special block every N lines | 5 | 0–50 (0 disables field specials) |
| Specials per trigger | 1 | 0–5 |
| Inventory capacity | 5 | 1–8 |
| Garbage on 2/3/4-line clears | on | on / off |

### What the options mean

- **Start level** — the level and falling speed applied to the first piece. It changes speed and the level display only; it does not add blocks. Higher is faster.
- **Starting garbage rows** — rows of mixed blocks, each with one gap, placed at the bottom of every player's field before the first piece. `0` starts with an empty field; more rows leave less room from the beginning.
- **Lines per level** — how many cleared lines a player must accumulate to gain a level. Lower values ramp the falling speed up sooner.
- **Level step per threshold** — how many levels are gained each time that threshold is reached. `1` is the normal ramp; higher values make each threshold much faster.
- **Special block every N lines** — after every N cumulative cleared lines, the game injects special blocks into that player's stack by replacing normal blocks. `0` disables injection. The counter carries over: clearing 6 lines with `N = 5` triggers once and leaves 1 line toward the next trigger.
- **Specials per trigger** — how many special blocks are injected each time that cadence fires. With the default 1, one block appears every 5 lines. `0` means the cadence is consumed but nothing is placed.
- **Inventory capacity** — the maximum number of collected special blocks you can hold at once. When the inventory is full, further collected specials are discarded. Every stored entry can be spent once.
- **Garbage on 2/3/4-line clears** — when on, clearing 2, 3, or 4 rows at once also pushes 1, 2, or 4 rows of garbage onto every other living player. When off, only special blocks send garbage.

### Round flow

- Clearing a full row removes it and collapses the rows above. Clearing several rows at once is worth more pressure when garbage attacks are on.
- Clear rows containing special blocks to collect them; clearing several such rows collects every special in them, up to the inventory capacity.
- A player is out when a new piece cannot enter the field, or when garbage pushes a row above the top.
- The last player still able to place pieces wins. Seven seconds after the round, the room returns to the lobby with the same options.
- A player dropped mid-round keeps their slot and board for 3 minutes and can reconnect with the same browser tab token; the board is frozen while disconnected. Tokens are tab-scoped ( `sessionStorage` ), so two tabs in one browser join as two different players, while reloading a tab keeps its player identity.

Specials:

| id | glyph | name | effect |
|---|---|---|---|
| 0 | A | Add Line | pushes one garbage line onto the target |
| 1 | C | Clear Line | shifts the target's field down and clears the top row |
| 2 | N | Nuke Field | empties the target's field |
| 3 | R | Clear Random | removes 10 random cells from the target |
| 4 | S | Switch Fields | swaps your field with the target's |
| 5 | B | Clear Specials | turns the target's special blocks into normal blocks |
| 6 | G | Block Gravity | compacts all columns downward on the target |
| 7 | Q | Blockquake | shifts every row horizontally by a random amount |
| 8 | O | Block Bomb | removes all bomb blocks and scatters their neighbours |

No teams, separate spectator connections, MIDI soundtrack, or shared-piece-sequence variant. Six block colours are used instead of five so each piece has its own hue.

## Protocol

Binary frames, first byte = message type. WebSocket frames are binary and unmasked server → client. The SSE fallback carries the same bytes base64-encoded in `event: bin` messages; client → server fallback uses `POST /api/input?sid=…` with the raw frame as the body.

Client → server:

| type | payload |
|---|---|
| `0x01` MOVE | `u8 action` : 0 left, 1 right, 2 rotate CW, 3 rotate CCW, 4 soft drop, 5 hard drop |
| `0x02` SPECIAL | `u8 special` , `u8 target slot` (1–6) |
| `0x03` START | — (host only, ≥2 connected, everyone ready, lobby phase) |
| `0x04` CHAT | UTF-8 text (≤240 chars) |
| `0x05` PING | 8-byte nonce, echoed by the server |
| `0x06` LEAVE | — |
| `0x07` RESYNC | — (server replies with a snapshot) |
| `0x08` OPTIONS | 8 × `u8` : start level, garbage rows, lines/level, level step, special every, special count, inventory cap, attacks (0/1) — host only, lobby only |
| `0x09` READY | `u8` 0/1 — lobby only |

Server → client:

| type | payload |
|---|---|
| `0x80` LOBBY | JSON: phase, host, winner, countdown ms, options, player list |
| `0x81` STATE | `u32 tick` , then 6 slot records |
| `0x82` BOARD | `u8 slot` , `u32 version` , 132 packed bytes |
| `0x83` EVENT | JSON: chat, line clears, attacks, special use, top-out, round over, errors |
| `0x84` SNAPSHOT | `u32 tick` , then 6 × ( `u8 present` , `u32 version` , 132 packed bytes) |
| `0x85` PONG | echoed nonce |
| `0x86` HELLO | JSON: your slot, reconnect token, game id |

`STATE` slot record (only when present):

```
u8 present | u8 alive | u8 level | u16 lines | u
i8 x | i8 y | u8 rot | u8 next | u8 n | n × u8 s
```

Board bytes pack two cells per byte, high nibble first. Cell values: 0 empty, 1–6 block colour, 7–15 special block ( `7 + special id` ). Boards are only broadcast when their version changes; `STATE` (~20–100 bytes) goes out at 22 Hz. Clients can request a snapshot after a tab wake-up or a missed frame.

Lobby JSON example:

```
{"game":"friday","phase":"lobby","host":1,"winne
 "options":{"initialLevel":1,"startingLines":0,"
            "specialEvery":5,"specialCount":1,"s
 "players":[{"id":1,"nick":"anne","wins":2,"conn
```

`phase` is `lobby` , `countdown` , `playing` , or `over` . `cd` is the countdown remaining in milliseconds. `ready` is the player's lobby readiness. `inGame` distinguishes players in the current round from players waiting for the next one.

## Transports

1. **WebSocket** `GET /api/ws?g=…&nick=…&token=…` — primary, lowest latency.
2. **SSE** `GET /api/sse?g=…&nick=…&token=…` (server → client) plus `POST /api/input?sid=…` (client → server). The stream starts with `event: hello` carrying the session id, then `event: bin` base64 frames.
3. Force the fallback with `…/g/<id>?transport=sse` ; the client otherwise tries WebSocket first and drops to SSE on failure, with exponential reconnect backoff.

`GET /api/status` is a small, uncached JSON endpoint:

```
{"maintenance":false,"activeGames":1}
```

The landing page polls it every 2 seconds to show how many rounds are in progress and whether the server is in maintenance.

All sockets are `TCP_NODELAY` ; state application is immediate and the rule engine runs on a 10 ms tick with 45 ms state broadcasts. Moves are applied on arrival, so input latency is one network RTT, not one tick.

## Shutdown and maintenance

Sending `SIGTERM` ( `kill -15` ) puts the server into maintenance mode:

- Existing rounds keep running and streaming.
- Pending countdowns are cancelled; new rooms, joins, and round starts are rejected with a maintenance message.
- The process exits with status 0 once no game with connected players remains.
- If no game is active at signal time, it exits immediately.

The landing page shows the maintenance state and the running-game count; invite links opened during maintenance show a “New games paused” overlay.

## Architecture

```
src/main.rs     listener, ticker thread
src/server.rs   routes, WebSocket + SSE + POST t
src/hub.rs      room registry, lobbies, countdow
src/game.rs     board, pieces, gravity, line cle
src/proto.rs    frame encoding/decoding and pack
src/http.rs     tiny HTTP/1.1 request parser and
src/ws.rs       WebSocket handshake, frame reade
src/shutdown.rs SIGTERM flag and maintenance mod
src/util.rs     SHA-1, base64, query parsing, sa
web/            single-page client (canvas, pred
tests/e2e.mjs   protocol integration test (Node
tests/maintenance.mjs  SIGTERM/maintenance integ
```

The server is server-authoritative: clients send inputs, never board states. Each room holds at most six player slots; the game state is mutated under one room mutex, which also serialises broadcasts. Rooms live in a `HashMap` and are dropped when empty (30 s in the lobby, 3 minutes mid-round to allow reconnects).

There are no third-party crates: HTTP, WebSocket framing (including masking and fragmentation), SHA-1/base64 for the handshake, and all game logic are in-tree.

## Testing

```
make test           # rules, specials, protocol
make e2e            # release build, starts a se
make maintenance    # SIGTERM: wait for games, r
make check          # all of the above
```

`make run` serves on the fixed address `127.0.0.1:8099`. For any other address run the binary directly with its flags. The equivalent manual e2e run:

```
cargo build --release
./target/release/cecris --bind 127.0.0.1 --port
BASE=http://127.0.0.1:8099 node tests/e2e.mjs
```

The e2e test drives two WebSocket clients through a full round to top-out, verifies state-frame rate, chat, board diffs, readiness gating and the 3-second countdown, host-edited options visible to a joiner and applied to the next round, an SSE player (including POSTed inputs and readiness), a parallel second room, and reconnect-with-token into a running game.

The maintenance test spawns its own server and verifies both shutdown paths: immediate exit when idle, and a running game that stays up, rejects new games with a maintenance message, keeps streaming to existing players, then exits after the round ends.

## Deployment

Run the binary behind a TLS-terminating reverse proxy; the client selects `wss` / `https` from `location.protocol`, and SSE needs buffering disabled:

```
example.com {
    reverse_proxy 127.0.0.1:8080
    flush_interval -1
}
```

SSE responses deliberately omit `Connection` and `Content-Length` and send `X-Accel-Buffering: no` with chunked transfer encoding. That combination is what keeps event streams flowing through Cloudflare Tunnel and nginx; do not re-add those headers in a proxy.

```
location /api/sse {
    proxy_pass http://127.0.0.1:8080;
    proxy_buffering off;
    proxy_cache off;
    proxy_read_timeout 1h;
}
location /api/ws {
    proxy_pass http://127.0.0.1:8080;
    proxy_read_timeout 1h;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
```

Static HTML, JS, and CSS are served with `Cache-Control: no-store`; the index links assets with a content hash (`/app.js?v=…`). After rebuilding and restarting the server, reload an already-open tab once — a tab still running an older client has no **Ready** button, so the server refuses to start its round.

There is no account system: the reconnect token in `sessionStorage` is a bearer capability for one player slot, and room ids are public.

## License

Copyright 2026 Aurélien DEHAY.

Licensed under the Apache License, Version 2.0. See LICENSE for the full text.

© 2026 GitHub, Inc.