Gruesøme’s Arcade — Game Integration Guide (v1.4)

Built: 2026-01-07
sig(b64)=YnVpbHQgYnkgZ3J1ZXPDuG1l

Scope
- How to embed a game in the Arcade shell
- How to request runs + submit run results
- How to attach metrics (generic + game-specific)
- How to stay compatible with Leaderboard v2 + Wallet payouts + anti-manipulation

This guide assumes:
- Static hosting under /public (Vercel outputDirectory = public)
- Embed adapter runtime at: /public/_lib/arcadeGameEmbedAdapter.js
- Run coordinator runtime at: /public/_lib/runCoordinator.js

1) Minimum Contract Between Game ↔ Arcade

The Arcade is the “parent” page. Your game runs inside an iframe and communicates via postMessage

Message types (standard)
- ARCADE:READY
- ARCADE:SYNC
- ARCADE:REQUEST_RUN
- ARCADE:RUN_GRANTED
- ARCADE:RUN_DENIED
- ARCADE:RUN_RESULT

Recommended payloads (high-level)

READY:
- gameId (string)
- version (string, optional)
- metricsVersion (string, optional)

SYNC:
- address (string)
- credits: { paid: number, promo: number }
- membership: string|null
- avatar: unknown|null

RUN_GRANTED:
- runId (string)
- runType (string)
- cost: { paidAC: number, promoAC: number }

RUN_RESULT:
```
- gameId (string)
- runId (string)
- durationMs (number)
- metricId (string)  // which metric is the “ranked” one for this run
- metricValue (number)
- metrics (object, optional) // extra metrics for telemetry/future
```

IMPORTANT: Some older game UIs expect a global getSync() helper.
To avoid “getSync is not defined” crashes, each embedded game SHOULD define:

```
  window.__ARCADE_SYNC = null;
  window.getSync = () => window.__ARCADE_SYNC;
```

…and update __ARCADE_SYNC whenever ARCADE:SYNC is received.

2) Recommended Game Bootstrap Template

In your game iframe:

1) Listen for messages from parent.
2) On load, post ARCADE:READY to parent.
3) On ARCADE:SYNC, store payload (and update UI).
4) When player wants a paid/promo run, send ARCADE:REQUEST_RUN.
5) When run ends, send ARCADE:RUN_RESULT with metricId + metricValue.

Template (drop-in skeleton):

```
  const TYPES = {
    READY: 'ARCADE:READY',
    SYNC: 'ARCADE:SYNC',
    REQUEST_RUN: 'ARCADE:REQUEST_RUN',
    RUN_GRANTED: 'ARCADE:RUN_GRANTED',
    RUN_DENIED: 'ARCADE:RUN_DENIED',
    RUN_RESULT: 'ARCADE:RUN_RESULT',
  };

  // Avoid legacy crashes
  window.__ARCADE_SYNC = null;
  window.getSync = () => window.__ARCADE_SYNC;

  function post(type, payload) {
    window.parent?.postMessage({ type, payload }, '*');
  }

  window.addEventListener('message', (ev) => {
    const msg = ev?.data;
    if (!msg || typeof msg !== 'object') return;

    if (msg.type === TYPES.SYNC) {
      window.__ARCADE_SYNC = msg.payload || null;
      // update HUD, credits display, etc.
      return;
    }

    if (msg.type === TYPES.RUN_GRANTED) {
      // start the run using msg.payload.runId + cost
      return;
    }

    if (msg.type === TYPES.RUN_DENIED) {
      // show reason
      return;
    }
  });
  // Announce readiness
  post(TYPES.READY, {
    gameId: 'yourGameId',
    version: '1.0.0',
    metricsVersion: 'v3.1'
  });

  // When player requests a run:
  function requestPaidRun() {
    post(TYPES.REQUEST_RUN, {
      gameId: 'yourGameId',
      desiredRunType: 'paid'
    });
  }

  // When run ends:
  function submitResult(runId, durationMs, metricId, metricValue, extraMetrics) {
    post(TYPES.RUN_RESULT, {
      gameId: 'yourGameId',
      runId,
      durationMs,
      metricId,
      metricValue,
      metrics: extraMetrics || null
    });
  }
```

■■■

3) Metrics: What To Send, What To Catalog

■■■

Every game should have:
- defaultMetric: the primary ranked metric for that game
- metrics[]: list of supported metrics for the game
- direction: “higher” or “lower” (per metric) for ranking
- clamp: optional bounds to reduce abuse/outliers

Generic metrics (work for almost any game)
- score (higher)
- durationMs (higher or lower depending on the game)
- wins / winRateBp (higher)
- streak (higher)
- attempts (higher; usually activity-side)
- accuracyBp (higher)
- completionBp (higher)
- damageDealt / damageTaken (context)
- deaths (lower)
- speedrunMs (lower)

Economy / spend-aware games (e.g., Storm the House 2)
- inRunSpendAC (lower is better; fairness guardrail)
- efficiency (higher; result per spend, e.g., waves per AC or kills per AC)
- waves / levelReached (higher)
- kills (higher)

Rules of thumb

- “Skill” metrics are harder to spam: accuracy, efficiency, winRate, speedrunMs.
- “Activity” metrics reflect engagement: attempts, durationMs, sessions, creditsSpent.

For robust Web3 payouts that are equitable (not equal):
- Combine skill and activity boards, then weight payouts via policy (server-side).
- Use spend caps + efficiency metrics so “buying power” does not dominate.

■■
4) Anti-manipulation Requirements (Minimum)
■■

Your game SHOULD:
- Only submit RUN_RESULT after a RUN_GRANTED with a valid runId.
- Include durationMs for every run.
- Avoid trusting localStorage as the source of truth for credits or membership.
- Prefer receiving SYNC from parent and displaying that state.

Arcade server SHOULD:
- Validate runId exists and is granted to that address + gameId
- Reject impossible metrics (e.g., negative, NaN, exceeding clamp)
- Rate-limit run requests per address/game
- (Optional) use deterministic replays or signed run grants for high-value games

■■
5) Troubleshooting (Common)
■■

Symptom: “getSync is not defined”
- Fix: define window.getSync() as shown above and set window.__ARCADE_SYNC on ARCADE:SYNC.

Symptom: “Failed to load module script… MIME type text/html”
- Usually: the server returned /index.html as a fallback for a missing .js file.
- Check that /public/games/<id>/<file>.js exists and that rewrites do not swallow static assets.

Symptom: “Too many active WebGL contexts”
- You are likely creating multiple THREE.WebGLRenderer instances without disposing.
- Ensure you call renderer.dispose() + renderer.forceContextLoss() on unmount.

End.