Comprehensive-Minimal GitHub Actions for a WASM C Game

Emscripten + WebGL2 + CMake Presets (repo-tailored)

November 3, 2025

Contents

1 Your repository layout (ASCII-safe) 3

2 Build strategy (tailored) 4

3 Workflow 1: Build & Test (Emscripten + CMake preset) 4

4 Workflow 2: CodeQL (C/C++) 5

5 Workflow 3: Dependency Review 6

6 Workflow 4: Deploy to GitHub Pages 6

7 Workflow 5: Release on tags (zip build-wasm) 7

8 Optional: add a Release preset for size/perf 8

9 Notes and repo-specific tips 9

10 Gaps & Risks Addressed & Implemented Updates 9

10.1 What changed at a glance 9

10.2 Minimal CMake/Ninja presets (drop-in) 10

11 Updated Workflows (drop-in, production-ready) 10

11.1 `.github/workflows/build.yml` — Build & Test (WASM) 10

11.2 `.github/workflows/codeql.yml` — CodeQL for C/C++ (manual build mode) . 11

11.3 `.github/workflows/dependency-review.yml` — PR Gate 12

11.4 `.github/workflows/pages.yml` — GitHub Pages (with QA & manual trigger) . 12

11.5 `.github/workflows/release.yml` — Tag-based release, wasm-opt, size budget . 14

12 Repository-wide defaults and docs 15

12.1 Job permission defaults (optional top-level) 15

12.2 README badges (copy/paste) 15

12.3 Notes on third-party pins 15

Executive Summary

Quick Start

Diagnostics workflow (optional)

Use a lightweight job to print versions and confirm Emscripten is installed:

```yaml
name: Diagnostics
on: [workflow_dispatch]
jobs:
  envcheck:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: mymindstorm/setup-emsdk@v14
      - run: |
          emcc --version
          cmake --version
          ninja --version || true
```

1. Ensure you have a `CMakePresets.json` with a `wasm-debug` configure preset that outputs to `build-wasm`.
2. Push a branch: the *Build & Test* workflow compiles with Emscripten and caches dependencies.
3. Open a PR: *Dependency Review* and *CodeQL* run as guardrails.
4. Merge to `main`: *Deploy to GitHub Pages* publishes a playable demo.
5. Create a tag (e.g., `v0.1.0`): *Release* packages WebAssembly assets for distribution.

This guide delivers a cohesive, production-ready CI/CD baseline for a C/WebGL2 game compiled to WebAssembly via Emscripten. It is tailored for repositories that use `CMakePresets.json` with a `wasm-debug` configure preset that outputs to `build-wasm`. The workflows form a minimal-yet-complete pipeline:

- **Build & Test**: Compile with Emscripten and your CMake preset, producing reproducible WebAssembly assets.
- **CodeQL (C/C++)**: Add SAST for C/C++ to catch correctness and security issues in CI.
- **Dependency Review**: Guard PRs by flagging risky transitive changes.
- **Deploy to GitHub Pages**: Publish the `build-wasm` output for playtesting.
- **Release on tags**: Package build artifacts as downloadable assets.

The result is a pragmatic path from commit to playable demo to versioned releases, with security checks and dependency risk controls built in.

Goal

Prerequisites

Before running the workflows, make sure the essentials are in place:

- **Emscripten SDK**: Installed and available on the runner via `emsdk`. Use the latest stable tools.
- **CMake presets**: A `CMakePresets.json` with a `wasm-debug` configure preset that generates to `build-wasm`.
- **CI resources**: GitHub Actions enabled with permissions to read repository contents and write Pages/Deployments.
- **Security scanning**: If you use CodeQL, ensure the repository or org has GitHub Advanced Security enabled.
- **Pages (optional)**: If you plan to publish a playable demo, enable GitHub Pages (build from GitHub Actions).

Ship a comprehensive-minimal CI/CD set for your WebAssembly (WASM) **C** game using Emscripten + WebGL2, matching your repository layout and CMake presets. We provide five production-ready GitHub Actions workflows:

- Build & Test (Emscripten + `wasm-debug` preset → `build-wasm`)
- CodeQL (C/C++)
- Dependency Review (PR guardrail)
- Deploy to GitHub Pages (publish `build-wasm`)
- Release on tags (zip `build-wasm` assets)

How to compile this PDF

```
latexmk -pdf -shell-escape main.tex
```

1 Your repository layout (ASCII-safe)

```
notoriousjayy-wasm/
|-- readme.md
|-- CMakeLists.txt
|-- CMakePresets.json
|-- package.json
|-- html_template/
|   `-- index.html
|-- include/
|   `-- testProject/
|       |-- module.h
|       `-- render.h
`-- src/
    |-- main.c
    |-- module.c
    `-- render.c
```

2 Build strategy (tailored)

Tunable Parameters

- **Preset name**: Switch from `wasm-debug` to `wasm-release` for optimized bundles.
- **Cache keys**: Include `runner.os`, preset name, `CMakeLists.txt` hash.
- **Artifact names**: Unify across workflows (e.g., `build-wasm`).
- **Pages directory**: If your web assets live under `web/`, set the publish path accordingly.
- **CodeQL query packs**: Add security-focused packs or narrow scope via `.codeql/config.yml`.

Your README and presets expect `emcmake` and the `wasm-debug` preset; the binary dir is `build-wasm`. The CMake project emits `index.html`, `index.js`, `index.wasm` (output name `index`) and will use `html_template/index.html` if present (it already includes the `{{{ SCRIPT }}}` placeholder). We mirror this in CI.

Local dev (reference)

```
# Emscripten SDK (once)
git clone https://github.com/emscripten-core/emsdk.git ~/emsdk
cd ~/emsdk && ./emsdk install latest && ./emsdk activate latest
source ~/emsdk/emsdk_env.sh

# Configure and build (matching README)
emcmake cmake --preset wasm-debug
cmake --build --preset wasm-debug

# Serve locally from the build dir
cmake --build --preset wasm-debug --target serve
# Open http://localhost:8000/
```

3 Workflow 1: Build & Test (Emscripten + CMake preset)

This step continues the pipeline, building on the outputs of earlier workflows.

```
# .github/workflows/build-wasm.yml
name: Build (WASM C)
on:
  push: { branches: [main] }
  pull_request: { branches: [main] }

permissions:
  contents: read

jobs:
  build:
    runs-on: ubuntu-latest
    env:
      EMSDK: ${{ runner.temp }}/emsdk

    steps:
      - uses: actions/checkout@v4

      - name: Install Emscripten
```

```
        shell: bash
        run: |
          git clone https://github.com/emscripten-core/emsdk.git "$EMSDK"
          "$EMSDK/emsdk" install latest
          "$EMSDK/emsdk" activate latest
          echo "EMSDK=$EMSDK" >> $GITHUB_ENV
          source "$EMSDK/emsdk_env.sh"
          emcc --version

      - name: Cache Emscripten build cache
        uses: actions/cache@v4
        with:
          path: |
            ~/.emscripten_cache
            ${{ env.EMSDK }}/upstream/emscripten/cache
          key: emsdk-${{ runner.os }}-latest

      - name: Configure (emcmake + preset wasm-debug)
        shell: bash
        run: |
          source "$EMSDK/emsdk_env.sh"
          emcmake cmake --preset wasm-debug

      - name: Build
        shell: bash
        run: cmake --build --preset wasm-debug -j2

      # No tests yet; keep as a future hook
      - name: Upload build artifacts
        uses: actions/upload-artifact@v4
        with:
          name: wasm-build
          path: |
            build-wasm/index.html
            build-wasm/index.js
            build-wasm/index.wasm
```

4 Workflow 2: CodeQL (C/C++)

This step continues the pipeline, building on the outputs of earlier workflows.

If `autobuild` cannot infer a build, reuse the emsdk + preset steps before analyze (shown commented).

```
# .github/workflows/codeql.yml
name: CodeQL (C/C++)
on:
  push: { branches: [main] }
  pull_request: { branches: [main] }
  schedule: [{ cron: "0 6 * * 1" }]

permissions:
  contents: read
  security-events: write

jobs:
  analyze:
    runs-on: ubuntu-latest
```

```
    steps:
      - uses: actions/checkout@v4

      - uses: github/codeql-action/init@v3
        with:
          languages: cpp  # covers C and C++

      - uses: github/codeql-action/autobuild@v3

      # If autobuild fails, uncomment and use your exact wasm build:
      # - name: Build (Emscripten, same as CI)
      #   shell: bash
      #   run: |
      #     EMSDK="$RUNNER_TEMP/emsdk"
      #     git clone https://github.com/emscripten-core/emsdk.git "$EMSDK"
      #     "$EMSDK/emsdk" install latest
      #     "$EMSDK/emsdk" activate latest
      #     source "$EMSDK/emsdk_env.sh"
      #     emcmake cmake --preset wasm-debug
      #     cmake --build --preset wasm-debug -j2

      - uses: github/codeql-action/analyze@v3
```

5 Workflow 3: Dependency Review

This step continues the pipeline, building on the outputs of earlier workflows.

```
# .github/workflows/dependency-review.yml
name: Dependency Review
on:
  pull_request:
    types: [opened, synchronize, reopened]

permissions:
  contents: read

jobs:
  review:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/dependency-review-action@v4
```

6 Workflow 4: Deploy to GitHub Pages

This step continues the pipeline, building on the outputs of earlier workflows.

We build with the wasm preset and publish `build-wasm` as the Pages artifact. Your custom HTML shell in `html_template/index.html` will be used automatically because CMake passes `-shell-file` when it exists.

```
# .github/workflows/pages.yml
name: Deploy (GitHub Pages)
on:
  push: { branches: [main] }

permissions:
```

```
  contents: read
  pages: write
  id-token: write

concurrency:
  group: "pages"
  cancel-in-progress: false

jobs:
  build:
    runs-on: ubuntu-latest
    env:
      EMSDK: ${{ runner.temp }}/emsdk

    steps:
      - uses: actions/checkout@v4

      - name: Install Emscripten
        shell: bash
        run: |
          git clone https://github.com/emscripten-core/emsdk.git "$EMSDK"
          "$EMSDK/emsdk" install latest
          "$EMSDK/emsdk" activate latest
          echo "EMSDK=$EMSDK" >> $GITHUB_ENV
          source "$EMSDK/emsdk_env.sh"

      - name: Configure & build (emcmake + preset)
        run: |
          emcmake cmake --preset wasm-debug
          cmake --build --preset wasm-debug -j2
          test -f build-wasm/index.html

      - name: Configure Pages
        uses: actions/configure-pages@v5

      - name: Upload artifact
        uses: actions/upload-pages-artifact@v3
        with:
          path: build-wasm

  deploy:
    needs: build
    runs-on: ubuntu-latest
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    steps:
      - id: deployment
        uses: actions/deploy-pages@v4
```

7 Workflow 5: Release on tags (zip build-wasm)

This step continues the pipeline, building on the outputs of earlier workflows.

```
# .github/workflows/release.yml
name: Release
on:
  push:
```

```yaml
    tags: ['v*.*.*']

permissions:
  contents: write

jobs:
  release:
    runs-on: ubuntu-latest
    env:
      EMSDK: ${{ runner.temp }}/emsdk

    steps:
      - uses: actions/checkout@v4

      - name: Install Emscripten
        shell: bash
        run: |
          git clone https://github.com/emscripten-core/emsdk.git "$EMSDK"
          "$EMSDK/emsdk" install latest
          "$EMSDK/emsdk" activate latest
          source "$EMSDK/emsdk_env.sh"

      - name: Build (emcmake + preset)
        run: |
          emcmake cmake --preset wasm-debug
          cmake --build --preset wasm-debug -j2
          (cd build-wasm && zip -r ../game-wasm.zip .)

      - name: Create GitHub Release
        uses: softprops/action-gh-release@v2
        with:
          files: game-wasm.zip
```

8 Optional: add a Release preset for size/perf

If you want a production-optimized build, extend `CMakePresets.json` with a `wasm-release` preset (still using `emcmake` in CI):

```json
{
  "version": 5,
  "cmakeMinimumRequired": { "major": 3, "minor": 20 },
  "configurePresets": [
    {
      "name": "wasm-debug",
      "displayName": "Emscripten / Debug",
      "generator": "Ninja",
      "binaryDir": "build-wasm",
      "cacheVariables": { "CMAKE_BUILD_TYPE": "Debug" }
    },
    {
      "name": "wasm-release",
      "displayName": "Emscripten / Release",
      "generator": "Ninja",
      "binaryDir": "build-wasm",
      "cacheVariables": {
        "CMAKE_BUILD_TYPE": "Release"
      }
    }
```

```
  ],
  "buildPresets": [
    { "name": "wasm-debug",   "configurePreset": "wasm-debug" },
    { "name": "wasm-release", "configurePreset": "wasm-release" }
  ]
}
```

Then update the workflows' build steps to use `wasm-release` for deploy and release jobs.

9 Notes and repo-specific tips

- **Custom shell**: your `html_template/index.html` already includes `{{{ SCRIPT }}}`, so the generated JS is injected correctly.
- **Exports**: CMake sets `-sEXPORTED_FUNCTIONS=['_main','_initWebGL','_startMainLoop','_myFuncti` and runtime methods `ccall,cwrap`, so DevTools calls like `Module.ccall('myFunction', ...)` work out of the box.
- **Artifacts**: Pages and Release jobs upload the entire `build-wasm` directory to preserve all outputs (`index.html/js/wasm`).
- **Cache**: caching `~/.emscripten_cache` and `upstream/emscripten/cache` speeds up builds.
- **Local serve target**: `cmake -build -preset wasm-debug -target serve` starts `python3 -m http.server` bound to `SERVE_HOST:SERVE_PORT` configured in CMake.

10 Gaps & Risks Addressed & Implemented Updates

10.1 What changed at a glance

- **Pinned toolchain and actions:** CMake/Ninja are used explicitly; Emscripten version is fixed; core Actions are pinned to immutable SHAs where practicable.
- **Deterministic builds:** a minimal `CMakePresets.json` enables `Ninja` and `CMAKE_EXPORT_COMPILE_COMMAN` for CodeQL.
- **Security hardening:** job/token permissions are least-privilege; third-party actions are minimized; Pages deploy has concurrency guards.
- **Dependency risk gate:** a PR-only dependency review job blocks high-severity advisories.
- **Release sanity:** `wasm-opt` optimization, a size budget check, and GitHub auto release notes.
- **Pages QA:** basic link check and a `workflow_dispatch` path for manual previews.
- **Headless smoke test:** quick Node-based load check for the generated JS glue.
- **Docs & badges:** drop-in Markdown badge snippet for CI/Pages/Release status.

10.2 Minimal CMake/Ninja presets (drop-in)

Place this in the repository root as `CMakePresets.json` to standardize local and CI builds and to emit `compile_commands.json` for CodeQL.

```
{
  "version": 3,
  "cmakeMinimumRequired": { "major": 3, "minor": 22, "patch": 0 },
  "configurePresets": [
    {
      "name": "wasm-release",
      "generator": "Ninja",
      "binaryDir": "build",
      "cacheVariables": {
        "CMAKE_BUILD_TYPE": "Release",
        "CMAKE_EXPORT_COMPILE_COMMANDS": "ON"
      }
    }
  ]
}
```

11 Updated Workflows (drop-in, production-ready)

11.1 `.github/workflows/build.yml` — Build & Test (WASM)

```
name: Build & Test (WASM)

on:
  push:
    branches: [ "main" ]
  pull_request:
  workflow_dispatch:
    inputs:
      emscripten:
        description: "emscripten/emsdk version (e.g., 3.1.59)"
        required: false

# sensible defaults and least privilege
permissions:
  contents: read

concurrency:
  group: build-${{ github.ref }}
  cancel-in-progress: true

jobs:
  build:
    runs-on: ubuntu-latest
    timeout-minutes: 15
    steps:
      - name: Checkout
        uses: actions/checkout@08c6903cd8c0fde910a37f88322edcfb5dd907a8 # v5
      - name: Restore emsdk cache
        uses: actions/cache@0057852bfaa89a56745cba8c7296529d2fc39830 # v4
```

```
        with:
          path: |
            ~/.cache/emscripten
            ~/.emscripten_cache
          key: ${{ runner.os }}-emsdk-${{ inputs.emscripten || '3.1.59' }}
      - name: Install build deps
        run: |
          sudo apt-get update
          sudo apt-get install -y ninja-build cmake
      - name: Setup Emscripten SDK
        # Pinning a third-party action ideally uses a full SHA; if avoiding
        ↪  third-party, do a manual install.
        # For simplicity and speed, we use the maintained setup action. You may
        ↪  replace with a manual script if desired.
        uses: mymindstorm/setup-emsdk@v14
        with:
          version: ${{ inputs.emscripten || '3.1.59' }}
      - name: Configure (Ninja + compile_commands.json)
        run: cmake -S . -B build -G Ninja -DCMAKE_BUILD_TYPE=Release
        ↪  -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
      - name: Build
        run: cmake --build build --parallel 2
      - name: Headless smoke test (Node load check)
        run: |
          set -eux
          test -f build/index.js || test -f build/main.js
          FILE=$( [ -f build/index.js ] && echo build/index.js || echo build/main.js )
          node -e "const fs=require('fs'); const
          ↪  s=fs.readFileSync(process.argv[1],'utf8'); if(!/Module/.test(s)) {
          ↪  throw new Error('Emscripten glue missing Module symbol'); }" "$FILE"
      - name: Upload artifact (bundle)
        uses: actions/upload-artifact@ea165f8d65b6e75b540449e92b4886f43607fa02 # v4
        with:
          name: wasm-bundle
          path: |
            build/**
          if-no-files-found: error
          retention-days: 7
```

11.2 .github/workflows/codeql.yml — CodeQL for C/C++ (manual build mode)

```
name: CodeQL (C/C++)

on:
  push:
    branches: [ "main" ]
  pull_request:
  schedule:
    - cron: "31 3 * * 0"

permissions:
  contents: read
  security-events: write
```

```yaml
jobs:
  analyze:
    runs-on: ubuntu-latest
    timeout-minutes: 30
    steps:
      - uses: actions/checkout@08c6903cd8c0fde910a37f88322edcfb5dd907a8 # v5
      - name: Install build deps
        run: |
          sudo apt-get update
          sudo apt-get install -y ninja-build cmake
      - name: Initialize CodeQL (C/C++, manual build)
        uses: github/codeql-action/init@v3
        with:
          languages: cpp
          queries: security-and-quality
          build-mode: manual
      - name: Configure (emit compile_commands.json)
        run: cmake -S . -B build -G Ninja -DCMAKE_BUILD_TYPE=Release
        ↪  -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
      - name: Build (no parallel to keep logs smaller)
        run: cmake --build build --parallel 1
      - name: Perform CodeQL Analysis
        uses: github/codeql-action/analyze@v3
```

11.3 .github/workflows/dependency-review.yml — PR Gate

```yaml
name: Dependency Review

on:
  pull_request:
    types: [opened, synchronize, reopened]

permissions:
  contents: read

jobs:
  review:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@08c6903cd8c0fde910a37f88322edcfb5dd907a8 # v5
      - name: Dependency Review (block high severity)
        uses: actions/dependency-review-action@v4
        with:
          fail-on-severity: high
          comment-summary-in-pr: true
```

11.4 .github/workflows/pages.yml — GitHub Pages (with QA & manual trigger)

```yaml
name: Pages

on:
  push:
```

```yaml
    branches: [ "main" ]
  workflow_dispatch:

permissions:
  contents: read
  pages: write
  id-token: write

concurrency:
  group: github-pages
  cancel-in-progress: true

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@08c6903cd8c0fde910a37f88322edcfb5dd907a8 # v5
      - name: Build site (static bundle)
        run: |
          cmake -S . -B build -G Ninja -DCMAKE_BUILD_TYPE=Release
          cmake --build build --parallel 2
          test -d build
      - name: Link sanity check
        run: |
          set -eux
          python3 - << 'PY'
          import os, sys, re
          bad = []
          for root, _, files in os.walk('build'):
              for f in files:
                  if f.endswith(('.html','.htm')):
                      p = os.path.join(root,f)
                      s = open(p,'r',errors='ignore').read()
                      for href in re.findall(r'href=[\'"](.*?)[\'"]', s):
                          if href.startswith('http'): continue
                          tgt = os.path.normpath(os.path.join(root, href))
                          if not os.path.exists(tgt):
                              bad.append((p, href))
          if bad:
              for p, h in bad[:20]:
                  print(f"Missing link: {p} -> {h}")
              sys.exit(1)
          PY
      - name: Upload Pages artifact (tarball)
        uses: actions/upload-pages-artifact@7b1f4a764d45c48632c6b24a0339c27f5614fb0b
        ↪  # v4.0.0
        with:
          path: build

  deploy:
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
```

```
    needs: build
    steps:
      - name: Deploy to GitHub Pages
        id: deployment
        # (Optionally pin this to a commit from actions/deploy-pages tags)
        uses: actions/deploy-pages@v4
```

11.5 .github/workflows/release.yml — Tag-based release, wasm-opt, size budget

```
name: Release (tag)

on:
  push:
    tags:
      - "v*.*.*"

permissions:
  contents: write

env:
  WASM_SIZE_BUDGET_KB: "1024"  # adjust as needed
  EM_VERSION: "3.1.59"

jobs:
  release:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@08c6903cd8c0fde910a37f88322edcfb5dd907a8 # v5
      - name: Install deps
        run: |
          sudo apt-get update
          sudo apt-get install -y ninja-build cmake
          npm i -g binaryen
      - name: Configure & build (Release)
        run: |
          cmake -S . -B build -G Ninja -DCMAKE_BUILD_TYPE=Release
          ↪  -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
          cmake --build build --parallel 2
      - name: Optimize wasm (wasm-opt -O3, strip)
        run: |
          set -eux
          WASM=$(ls build/*.wasm | head -n1)
          wasm-opt -O3 --strip-debug --strip-dwarf -o "${WASM%.wasm}.opt.wasm" "$WASM"
          mv "${WASM%.wasm}.opt.wasm" "$WASM"
      - name: Size budget check
        run: |
          set -eux
          WASM=$(ls build/*.wasm | head -n1)
          sz=$(du -k "$WASM" | cut -f1)
          echo "WASM size (KB): $sz"
          if [ "$sz" -gt "${WASM_SIZE_BUDGET_KB}" ]; then
            echo "::error title=Size budget exceeded::${sz} KB >
            ↪  ${WASM_SIZE_BUDGET_KB} KB"
```

```
        exit 1
      fi
  - name: Upload build as artifact
    uses: actions/upload-artifact@ea165f8d65b6e75b540449e92b4886f43607fa02 # v4
    with:
      name: release-bits
      path: build/**
      retention-days: 5
  - name: Create GitHub Release (notes generated)
    env:
      GH_TOKEN: ${{ secrets.GITHUB_TOKEN }}
    run: |
      set -eux
      TAG="${GITHUB_REF_NAME}"
      gh release create "$TAG" build/* --generate-notes --title "$TAG"
```

12 Repository-wide defaults and docs

12.1 Job permission defaults (optional top-level)

Add to each workflow if you want to guarantee the most restrictive default and override per job as needed:

```
permissions:
  contents: read
```

12.2 README badges (copy/paste)

```
![Build](https://github.com/<OWNER>/<REPO>/actions/workflows/build.yml/badge.svg)
![CodeQL](https://github.com/<OWNER>/<REPO>/actions/workflows/codeql.yml/badge.svg)
![Pages](https://github.com/<OWNER>/<REPO>/actions/workflows/pages.yml/badge.svg)
![Release](https://github.com/<OWNER>/<REPO>/actions/workflows/release.yml/badge.svg)
```

12.3 Notes on third-party pins

Where a full commit SHA is not provided above (e.g., `mymindstorm/setup-emsdk`), prefer pinning to an immutable SHA for supply-chain safety or replace with an inline installation script.

Quality Gates & Branch Protection

To make CI results *enforced*, enable branch protection on `main` (Settings → Branches):

- Require status checks to pass before merging: include `Build & Test`, `CodeQL`, and `Dependency Review`.
- Require branch to be up to date before merging (optional, increases merge latency but avoids drift).
- Restrict who can push to `main`; favor PR-based changes.
- (Pages) Use environments with required reviewers if you want controlled promotions.

Risks & Mitigations

- **Stale caches**: Emscripten or CMake cache may cause odd build diffs. *Mitigation*: Include cache keys derived from the preset and a hash of `CMakeLists.txt`; provide a manual cache-bust input.
- **Insufficient tests**: WebAssembly builds pass but gameplay regresses. *Mitigation*: Add headless smoke tests (e.g., `wasm3` or minimal browser run) and validate key exported functions.
- **Large bundle size**: Slow Pages loads and timeouts. *Mitigation*: Enable `-O3` for release, compress (`.gz/.br`) on publish, and serve cache headers.
- **CodeQL noise**: First runs can flag legacy or third-party code. *Mitigation*: Triage to *dismiss with reason* or suppress safely; add a `.codeql/config.yml` to focus scope.
- **Supply-chain drift**: Dependency changes sneak in via submodules or vendored libs. *Mitigation*: Lock versions; review diffs; use Dependabot & Dependency Review.
- **Pages misconfiguration**: 404s or wrong directory published. *Mitigation*: Confirm `build-wasm` path; verify environment and `pages` permissions.

Common Pitfalls & Fixes

- **LaTeX Unicode tree chars**: Replace box-drawing characters with ASCII or `\texttt` blocks.
- **Minted & Pygments frozencache errors**: Build with `-shell-escape`; avoid `frozencache` unless the styles exist; run `latexmk` without `outputdir` or configure `minted` accordingly.
- **CodeQL not running**: Ensure GitHub Advanced Security is enabled at org/repo and the CodeQL workflow has correct permissions.
- **Pages publish fails**: Confirm the `pages` and `id-token` permissions and the correct `artifact` path.
- **Emscripten not found**: Ensure the setup step calls `emsdk` and `source "$EMSDK/emsdk_env.sh"` before invoking `emcmake/emcc`.

Wrap-up & next steps

You now have a cohesive, repo-tailored CI/CD path: build and test with Emscripten, scan with CodeQL, guard dependency risk, publish a playable demo, and ship tagged releases. From here, consider small, incremental upgrades:

- Add cache keys per preset/commit to speed builds even further.
- Extend tests with headless `wasm3` or browser-based smoke checks.
- Gate Pages deploys behind checks (e.g., `required_status_checks`) for stricter quality bars.
- Include Lighthouse (Pages) or Playwright runs for basic rendering validation.