Comprehensive-Minimal GitHub Actions for a WASM C++ Game

Emscripten + CMake Presets + Conan/vcpkg + SDL Flags

November 3, 2025

Goal

Stand up a comprehensive-minimal CI/CD set for a WebAssembly (WASM) C++ game using Emscripten. This doc gives you:

- A tailored repo layout, CMakePresets and CMakeLists tuned for Emscripten + SDL.
- Optional integration with Conan or vcpkg.
- Five production-ready GitHub Actions workflows: Build & Test, CodeQL, Dependency Review, Pages Deploy, Release.

How to compile this PDF (for your notes)

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

1 Repository Layout (ASCII-safe)

Below assumes a common but flexible layout. Adjust names if your repo differs.

```
.
|-- CMakeLists.txt
|-- CMakePresets.json
|-- src/
|   |-- main.cpp
|   |-- game/
|   |   |-- engine.cpp
|   |   `-- engine.hpp
|-- include/
|   `-- game/
|       `-- public.hpp
|-- assets/
|   |-- sprites/
|   |-- audio/
|   `-- index.html  (optional: a custom shell)
|-- external/       (optional: headers/libs you vendor yourself)
|-- conanfile.txt   (optional)
|-- vcpkg.json      (optional)
|-- .github/
|   `-- workflows/  (the 5 workflows in this doc)
`-- out/            (build outputs; we'll target out/wasm or out/site)
```

2 Emscripten-aware CMake Presets

Use CMakePresets.json to normalize local builds and CI. The preset below:

- Configures a wasm-release preset using Emscripten's toolchain.
- Builds with Ninja.
- Routes binaries/artifacts to out/wasm and static site to out/site.
- Enables SDL2 and typical Emscripten flags for browser games.

```
{
  "version": 6,
  "cmakeMinimumRequired": { "major": 3, "minor": 25, "patch": 0 },
  "configurePresets": [
    {
      "name": "wasm-release",
      "displayName": "WASM (Emscripten) - Release",
      "generator": "Ninja",
      "binaryDir": "${sourceDir}/out/wasm",
      "cacheVariables": {
        "CMAKE_BUILD_TYPE": "Release",
        "CMAKE_TOOLCHAIN_FILE":
        ↪  "${env:EMSDK}/upstream/emscripten/cmake/Modules/Platform/Emscripten.cmake",
        "CMAKE_RUNTIME_OUTPUT_DIRECTORY": "${sourceDir}/out/wasm/bin",
        "CMAKE_LIBRARY_OUTPUT_DIRECTORY": "${sourceDir}/out/wasm/lib",
        "CMAKE_ARCHIVE_OUTPUT_DIRECTORY": "${sourceDir}/out/wasm/lib",
        "GAME_ASSETS_DIR": "${sourceDir}/assets",
        "GAME_SITE_DIR": "${sourceDir}/out/site",
        "GAME_ENABLE_SDL": "ON",
        "GAME_SDL_IMAGE_FORMATS": "png,jpg",
        "GAME_ASSUME_WASM": "ON"
      },
      "environment": {
        "EMSDK": "$env{EMSDK}"
      }
    }
  ],
  "buildPresets": [
    {
      "name": "wasm-release",
      "configurePreset": "wasm-release",
      "jobs": 4
    }
  ],
  "testPresets": [
    {
      "name": "wasm-ctest",
      "configurePreset": "wasm-release",
      "output": { "outputOnFailure": true }
    }
  ]
}
```

3 CMakeLists.txt (WASM-friendly, SDL-enabled)

The CMake below detects Emscripten, applies sane defaults, and copies assets to your site output. It uses target_link_options for Emscripten flags (SDL, memory growth, preloading assets).

```cmake
cmake_minimum_required(VERSION 3.25)
project(WasmGame LANGUAGES C CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Inputs from presets (with defaults)
set(GAME_ASSETS_DIR        "${GAME_ASSETS_DIR}"        CACHE PATH "Assets dir")
set(GAME_SITE_DIR          "${GAME_SITE_DIR}"          CACHE PATH "Site output dir")
set(GAME_ENABLE_SDL        "${GAME_ENABLE_SDL}"        CACHE BOOL "Enable SDL2")
set(GAME_SDL_IMAGE_FORMATS "${GAME_SDL_IMAGE_FORMATS}" CACHE STRING "SDL_image formats csv")
set(GAME_ASSUME_WASM       "${GAME_ASSUME_WASM}"       CACHE BOOL "Assume Emscripten build")

file(GLOB_RECURSE GAME_SRC CONFIGURE_DEPENDS
     "${CMAKE_SOURCE_DIR}/src/*.cpp" "${CMAKE_SOURCE_DIR}/src/*.c")
add_executable(game ${GAME_SRC})

target_include_directories(game PRIVATE
  "${CMAKE_SOURCE_DIR}/include"
)

# Detect Emscripten / WASM
if(GAME_ASSUME_WASM OR "${CMAKE_SYSTEM_NAME}" STREQUAL "Emscripten")
  message(STATUS "Configuring Emscripten/WASM build")

  # Optimize for size (typical for web games)
  target_compile_options(game PRIVATE -O3)
  target_link_options(game PRIVATE -O3)

  # Common useful runtime flags
  target_link_options(game PRIVATE
    "-sALLOW_MEMORY_GROWTH=1"
    "-sMODULARIZE=1"
    "-sENVIRONMENT=web"
    "-sEXPORTED_RUNTIME_METHODS=['ccall','cwrap','FS']"
    "-sASSERTIONS=0"
    "-sFILESYSTEM=1"
  )

  # SDL2 (and SDL_image/mixer) via Emscripten ports
  if(GAME_ENABLE_SDL)
    target_link_options(game PRIVATE "-sUSE_SDL=2")
    if(GAME_SDL_IMAGE_FORMATS)
      # Example: png,jpg
      string(REPLACE "," ";" SDL_IMG_LIST "${GAME_SDL_IMAGE_FORMATS}")
      set(SDL_IMG_JSON "[")
      foreach(fmt IN LISTS SDL_IMG_LIST)
        if(SDL_IMG_JSON STREQUAL "[")
          set(SDL_IMG_JSON "\"${fmt}\"")
        else()
          set(SDL_IMG_JSON "${SDL_IMG_JSON},\"${fmt}\"")
        endif()
      endforeach()
      set(SDL_IMG_JSON "[${SDL_IMG_JSON}]")
      target_link_options(game PRIVATE "-sUSE_SDL_IMAGE=2"
      ↪  "-sSDL2_IMAGE_FORMATS=${SDL_IMG_JSON}")
    endif()
    # Uncomment to enable SDL_mixer:
    # target_link_options(game PRIVATE "-sUSE_SDL_MIXER=2")
  endif()

  # Produce .html launcher alongside .wasm/.js
  set_target_properties(game PROPERTIES SUFFIX ".html")

  # Preload assets into virtual FS (so fetches work when served statically)
  if(EXISTS "${GAME_ASSETS_DIR}")
    target_link_options(game PRIVATE
      "--preload-file" "${GAME_ASSETS_DIR}@/assets"
    )
  endif()

  # Copy an index.html shell if you keep one in assets/
  add_custom_command(TARGET game POST_BUILD
    COMMAND ${CMAKE_COMMAND} -E make_directory "${GAME_SITE_DIR}"
    COMMAND ${CMAKE_COMMAND} -E copy_if_different
            "$<TARGET_FILE_DIR:game>/game.html" "${GAME_SITE_DIR}/index.html"
    COMMAND ${CMAKE_COMMAND} -E copy_directory
            "${CMAKE_RUNTIME_OUTPUT_DIRECTORY}" "${GAME_SITE_DIR}"
  )
endif()
```

Minimal src/main.cpp

```
#include <cstdio>
#ifdef __EMSCRIPTEN__
  #include <emscripten.h>
#endif

int main() {
  std::puts("Hello WASM world!");
  #ifdef __EMSCRIPTEN__
    // Game loop would go here; keep process alive if needed.
    // emscripten_set_main_loop_arg(...);
  #endif
  return 0;
}
```

4 Optional: Conan (native) and vcpkg (including wasm)

Conan (handy for native dev; Emscripten support varies per package)

```
[requires]
# Example (native dev only): sdl/2.30.0

[generators]
CMakeDeps
CMakeToolchain
```

Typical native configure (not used for wasm CI):

```
conan profile detect --force
conan install . --output-folder=out/native --build=missing
cmake --preset default        # your native preset
cmake --build --preset default -j
```

vcpkg (works with Emscripten triplet)

```
{
  "name": "wasm-game",
  "version-string": "0.1.0",
  "builtin-baseline": "b8b6a3a44c2c1f1a0b5a8c78e8e5bfeccb8e1c3e",
  "dependencies": [
    "sdl2",
    "sdl2-image",
    "sdl2-mixer"
  ]
}
```

Example triplet vcpkg/tri/wasm32-emscripten.cmake:

```
set(VCPKG_TARGET_ARCHITECTURE wasm32)
set(VCPKG_CRT_LINKAGE dynamic)
set(VCPKG_LIBRARY_LINKAGE dynamic)
set(VCPKG_CMAKE_SYSTEM_NAME Emscripten)
```

Configure with vcpkg toolchain (optional):

```
cmake -S . -B out/wasm \
  -DCMAKE_TOOLCHAIN_FILE=$VCPKG_ROOT/scripts/buildsystems/vcpkg.cmake \
  -DVCPKG_TARGET_TRIPLET=wasm32-emscripten
```

5 Workflow #1 – Build & Test (Emscripten + CMake Presets)

Key points:

- Installs emsdk, exports EMSDK and sources env.
- Uses your wasm-release preset (Section 2).
- Uploads compiled artifacts.
- Caches Emscripten cache for speed.

```
# .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

      - 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 (CMakePreset: wasm-release)
        shell: bash
        run: |
          source "$EMSDK/emsdk_env.sh"
          cmake --preset wasm-release

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

      - name: (Optional) ctest
        shell: bash
        run: ctest --preset wasm-ctest || true

      - name: Upload build artifacts
        uses: actions/upload-artifact@v4
        with:
          name: wasm-build
          path: |
            out/wasm/**/*.wasm
            out/wasm/**/*.js
            out/wasm/**/*.data
            out/wasm/**/*.html
```

6 Workflow #2 – CodeQL (C/C++)

If autobuild fails, reuse your Emscripten install + CMake preset before analyze.

```
# .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

      - uses: github/codeql-action/autobuild@v3
        # If this fails, replace with:
        # - run: |
        #     git clone https://github.com/emscripten-core/emsdk.git "$RUNNER_TEMP/emsdk"
        #     "$RUNNER_TEMP/emsdk/emsdk" install latest
        #     "$RUNNER_TEMP/emsdk/emsdk" activate latest
        #     source "$RUNNER_TEMP/emsdk/emsdk_env.sh"
        #     cmake --preset wasm-release
        #     cmake --build --preset wasm-release -j2

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

7 Workflow #3 – Dependency Review

Guards PRs that introduce vulnerable dependencies (helpful if you add third-party code or web assets).

```
# .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
```

8 Workflow #4 – Deploy to GitHub Pages

Builds with your preset and publishes `out/site` to Pages.

```
# .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 (CMake Preset)
        run: |
          cmake --preset wasm-release
          cmake --build --preset wasm-release -j2
          mkdir -p out/site
          find out/wasm -type f \( -name "*.wasm" -o -name "*.js" -o -name "*.data" -o -name
          ↪  "*.html" \) -exec cp -t out/site {} +

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

      - name: Upload artifact
        uses: actions/upload-pages-artifact@v3
        with:
          path: out/site

  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
```

9 Workflow #5 – Release (Tag -> Assets)

On `v*.*.*` tags, builds and attaches a zip. Uses a dedicated action for reliability.

```
# .github/workflows/release.yml
name: Release
on:
  push:
    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 (CMake Preset)
        run: |
          cmake --preset wasm-release
          cmake --build --preset wasm-release -j2
          mkdir -p release
          find out/wasm -type f \( -name "*.wasm" -o -name "*.js" -o -name "*.data" -o -name
          ↪  "*.html" \) -exec cp -t release {} +
          (cd release && zip -r game-wasm.zip .)

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

10 Local Build Cheats

Build locally with preset

```
# 1) Install and activate emsdk (once)
git clone https://github.com/emscripten-core/emsdk.git ~/emsdk
~/emsdk/emsdk install latest
~/emsdk/emsdk activate latest
source ~/emsdk/emsdk_env.sh

# 2) Configure & build
cmake --preset wasm-release
cmake --build --preset wasm-release -j

# 3) Serve (any static server). Example:
python3 -m http.server --directory out/site 8080
```

11 Notes & Tips

- Caching: We cache both `~/.emscripten_cache` and `EMSDK/upstream/emscripten/cache`.
- Assets: Using `-preload-file` ensures your assets load from a static host like Pages.
- SDL: If you need image formats beyond PNG/JPG, expand `GAME_SDL_IMAGE_FORMATS`.
- Performance: Start with `-O3` and `-sALLOW_MEMORY_GROWTH=1`. Consider LTO if your codebase benefits.
- CodeQL: For complex builds, run your exact Emscripten steps before `analyze`.