GB STUDIO GUIDE

How to Build a Super Mario-Inspired Game and Serve It on gb.spqw.net
Prepared on March 28, 2026

OVERVIEW

This guide covers the practical technical work needed to make a Game Boy platformer in GB Studio that captures the feel of early Mario games without copying Nintendo IP, then deploy it as a static web build on gb.spqw.net.

LEGAL LINE

- Copy design grammar, pacing, and readability, not protected expression.
- Do not reuse Mario, Luigi, Goombas, Koopas, pipes, question blocks, mushrooms, coins, Nintendo music, or direct level remakes.
- Make your own hero, enemies, world themes, pickup economy, and UI language.

1. WHAT GB STUDIO IS GOOD AT

- Short, handcrafted stages split into multiple scenes.
- Readable movement, clean jump arcs, and simple enemy timing.
- Event-driven gameplay systems such as checkpoints, pickups, and exits.
- Strong visual constraints that force compact, modular level art.

2. KEY TECHNICAL CONSTRAINTS

- Backgrounds: PNGs in assets/backgrounds, four colors, dimensions in multiples of 8 pixels, minimum 160x144, maximum width or height 2040, maximum 192 unique 8x8 tiles.
- Sprites: PNGs in assets/sprites, four-color sprite palette, built on 16x16 frames. A common animated actor strip is 96x16.
- Web export: Export Web produces static files in build/web.
- Custom Events: use them for reusable logic like pickups, damage, respawn, enemy defeat, and stage finish.
- Engine eject exists, but only use it if normal GB Studio workflows are not enough. Most projects should stay inside stock GB Studio.

3. RECOMMENDED FIRST VERSION

- One world.
- Three stages.
- One hero.
- Two enemy types.
- One collectible.
- One checkpoint system.
- One clear end goal.

Stage plan:

1. Stage 1 teaches movement and safe jumps.
2. Stage 2 adds tighter spacing, pits, and stronger enemy timing.
3. Stage 3 tests mastery and ends with a clear exit.

4. ASSET PIPELINE

Backgrounds

- Draw on an 8x8 grid from the beginning.
- Build a tile kit first: floor edge, floor fill, wall, platform, ladder, hazard, decor, and exit marker.
- Audit unique tile count early so you stay well under the 192-tile limit.
- Reuse motifs heavily for clouds, hills, caves, and machines.

Sprites
- Player: start with a simple walk and jump set.
- Enemy A: walker with minimal animation.
- Enemy B: hopper or stationary hazard.
- Pickup: static 16x16 if possible.

UI
- Customize ascii.png and frame.png after gameplay works.
- Keep text short. Platformers should not stop the player often.

5. SCENE CONSTRUCTION
- Treat each scene as one gameplay chunk, not the entire level.
- Scene 1: intro and teaching space.
- Scene 2: mid-level challenge and checkpoint.
- Scene 3: final push and exit.
- Split long levels into scenes connected by edge transitions, doors, or checkpoint warps.

6. COLLISION AND READABILITY
- Mark floors and walls first.
- Leave decorative background tiles non-solid.
- Use a distinct visual language for hazards such as spikes, acid, crushers, voids, or electric surfaces.
- Retest jump clearance constantly. One-tile mistakes feel much worse at Game Boy scale.

7. EVENT LOGIC TO BUILD EARLY
Create reusable Custom Events for:
- CollectPickup: increment score or currency, play SFX, hide actor, mark collected.
- DamagePlayer: reduce health or lives, apply knockback or respawn, create an invulnerability window.
- SetCheckpoint: store current scene and coordinates.
- RespawnPlayer: move the player to the checkpoint and reset temporary state.
- DefeatEnemy: play stomp or hit sequence, award points, hide enemy.
- FinishStage: lock controls, play finish cue, then transition to the next stage or score scene.

Variables to define on day one:
- v_lives
- v_score
- v_pickups
- v_checkpoint_scene

- v_checkpoint_x
- v_checkpoint_y
- v_stage_unlocked

8. ENEMY DESIGN
Start with three behavior families:
1. Walker: patrol and reverse.
2. Hopper: idle, jump, land, repeat.
3. Hazard: static danger, crusher, or projectile lane.

These three patterns are enough for a strong first game if the geometry is varied well.

9. MOVEMENT FEEL CHECKLIST
- The standard jump clears a two-tile obstacle reliably.
- Landing does not slide unpredictably.
- Enemy hits feel fair and readable.
- The camera does not disorient the player between scenes.
- The game feels good in one flat test room before content expansion begins.

10. DEVELOPMENT ORDER
1. Graybox one test room.
2. Tune movement.
3. Add one pickup and one enemy.
4. Add death and respawn.
5. Add checkpoint.
6. Add level exit.
7. Replace placeholders with final art.
8. Add music and sound last.

11. AUDIO
- Use short jump, pickup, damage, and finish sounds.
- Keep loops concise and memorable.
- Prioritize clarity over complexity. The player should always understand when they jumped, got hit, collected something, or finished a stage.

12. PERFORMANCE AND SCOPE DISCIPLINE
- Limit animated actors per scene.
- Reuse background tiles aggressively.
- Keep decoration secondary to gameplay readability.
- Prefer repeated enemy logic over bespoke one-off scripts.
- Split long levels into scenes instead of forcing one giant map.

13. EXPORTING FOR THE WEB
1. Use Export Web in GB Studio.
2. Find the generated files in build/web.
3. Verify index.html and assets load locally in a browser.
4. Deploy the exported static files to your site repository.

The official docs describe the web build as a static HTML5 export, so no backend is required unless you add extra services such as analytics.

14. SERVING THE GAME ON gb.spqw.net

Your current gb.spqw.net setup is already a static nginx site, which is a good match for GB Studio web exports.

You have two clean options:

- Replace the site root if the entire domain should become the game.
- Host the game under a subpath like /play/v1/ if you want a landing page plus the playable build.

Recommended structure for this repo:

1. Keep a landing page at the root.
2. Put each GB Studio export in a versioned folder such as /play/v1/.
3. Point the homepage Play button to the latest stable export.
4. Commit the static files to the git repository behind the Coolify app.
5. Redeploy the Coolify application after each release.

15. PRACTICAL DEPLOYMENT WORKFLOW

1. Finish the GB Studio game and run Export Web.
2. Copy the build/web output into this repo, usually under play/v1/.
3. Update index.html to link to the new export.
4. Commit and push to origin/master for spqw/gb.git.
5. Trigger a Coolify redeploy for the gb-spqw-net application.
6. Smoke-test the result on desktop and mobile at https://gb.spqw.net.

16. FIRST 7-DAY BUILD PLAN

Day 1: define theme, hero, enemy silhouettes, tile kit, and naming.
Day 2: graybox a test scene and tune movement.
Day 3: add enemy, pickup, death, and respawn.
Day 4: add checkpoint and level finish.
Day 5: replace placeholders with final art.
Day 6: add music, SFX, and script cleanup.
Day 7: export web build, stage it into the site repo, deploy, and test.

17. FINAL CHECKLIST

- No Nintendo art, names, music, character designs, or level copies remain.
- The player always understands where to go.
- Hazards and jumps are fair.
- Checkpoint state survives scene transitions.
- The web build loads on desktop and mobile.
- The homepage explains controls and links to the playable build.

SOURCES

Prepared against official GB Studio documentation at develop.gbstudio.dev, especially the sections covering backgrounds, sprites, scenes, custom events, settings, building your game, and engine eject.