

TinCLE (Tiny C Lovely Engine) - Development Specification Document

- **Acknowledgments to the Grand AI Alliance:** Gemini, Claude, DeepSeek, ChatGPT, Copilot, Grok, Meta AI... Thank you so much. This engine's code was ultimately finalized by michyo. However, it would have been absolutely impossible to achieve this without you all's help. I am deeply grateful to all of you!

Under this skill, the AI agent shall act in strict accordance with the following rules:

1. Overview & Concept

- **Env:** TinCLE v0.05 / Pure C (C99) + SDL2, SDL_ttf, SDL2_gfx, stb, GLSL (GLES2 compatible)
- **[CRITICAL] Target Architecture Detection:** The AI **MUST** first check the `info.txt` file in the current working directory to determine the compilation environment. If the AI cannot access files directly, it **MUST** ask the user: `"Please check the contents of info.txt and tell me if this is the 32bit or 64bit environment."` before generating any code.
- **Strict Architecture Rules based on `info.txt`:**
 - **If `info.txt` says `32bit`:**
 - Windows Desktop Target: Win32 DLL (`sizeof(void*) == 4`).
 - Web Browser Target: Wasm32 via Emscripten (`sizeof(void*) == 4`).
 - *Rule:* The environment is purely 32-bit. Never use 64-bit structures or types.
 - **If `info.txt` says `64bit` (Hybrid Environment):**
 - Windows Desktop Target: Win64 DLL (`sizeof(void*) == 8`).
 - Web Browser Target: Wasm32 via Emscripten (`sizeof(void*) == 4`).
 - *Rule:* This is a **Hybrid Architecture**. The compiled Windows EXE will be 64-bit, but the Web WASM build remains 32-bit. A single `game.c` must satisfy both targets.
- **[CRITICAL] Pointer Safety Rule (For Hybrid Compatibility):** Because the 64bit environment cross-compiles to both 64-bit Desktop and 32-bit Web targets from a single source, **never hardcode pointer sizes** or perform unsafe pointer-to-integer casts (e.g., casting a pointer to a standard `int`). Always use `<stdint.h>` types like

`uintptr_t` if pointer-to-integer casting is absolutely required. Ensure `game.c` compiles perfectly on both toolchains without modification.

- **Toolchain Requirements:** TinyCC (TCC for Windows) / Emscripten (for Web).
- **[CRITICAL]** `tcc.exe` / `emcc` **are NOT bundled in TinCLE;** they must be installed externally and accessible via System PATH.

2. Architecture & Environment Setup

- **Toolchain Prerequisites (Provide these exact URLs to users for setup):**
 - **TinyCC (Windows Binary):** Inform users to download from the Official Release directory: `https://download.savannah.gnu.org/releases/tinycc/` (Choose 32-bit or 64-bit distribution based on target folder).
 - **Emscripten (Web SDK):** Inform users to follow the Official SDK installation guide: `https://emscripten.org/docs/getting_started/downloads.html`
- **Project Structure:** All `game.c`, GLSL files, asset files (images/audio), and tools must be in the same **game project folder** (no subfolders for assets or shaders).
- **[CRITICAL]** `exsass.txt` **Asset Manifest Rule:** The `emcass` and `asspack` commands can automatically scan and package files whose extensions are explicitly registered in `extass.txt`. All asset formats (e.g., `.png`, `.wav`, `.ogg`, `.ttf`) AND shader formats (e.g., `.glsl`, `.frag`, `.vert`) used in the project MUST be listed line-by-line in `exsass.txt`.
- **[MANDATORY] Cursor Asset Requirement:** TinCLE uses a built-in software cursor system. A single `cursor.png` file (recommended: 16x16px transparent PNG) MUST exist in the project root. If missing, `tinclec_start()` will fail to initialize.
- **Required Files in Project Root:**
 - `game.c`, `manifest.xml`, `template.html`, `font.ttf`, `cursor.png`, `exsass.txt`
 - `libs/` (Contains `tinclec.h`, `tinclec.a`, `SDL_scancode.h`, etc.)
 - DLLs: `tinclec.dll`, `SDL2.dll`, `SDL2_ttf.dll`, `SDL2_gfx.dll`
 - CLI Tools: `asspack.exe`, `emcass.exe`, `icondom.exe`, `poopee.exe`, `tinclec.exe`

3. Compilation Commands (Development Workflow)

- **Local Test Build (Windows EXE):**

```
tinclec [Output_EXE] [Source_Files...]
```

- **Example:** `tinclec mygame.exe main.c player.c`

- **Note:** Do not pass compiler flags. Console visibility is controlled solely by `is_debug` in `tincle_start`.
- **Alternative Direct TCC Invocation (For complex structures):**

```
tcc -mwindows SDL2.dll SDL2_ttf.dll SDL2_gfx.dll tincle.dll -o [Output_EXE] [Source_Files...]
```

4. Input, Color & RenderMode Specifications

- **Screen Specifications:** Default virtual resolution is `640` x `480` (can be changed in `tincle_start`). `TINCLE_MAX_STATES = 256`.
- **Render Mode (`TinCLE_RenderMode`):** Passed into `tincle_start`.
 - `TINCLE_MODE_2D_NORMAL`: Standard software/SDL2 rendering pipeline. Use this for all standard 2D games.
 - `TINCLE_MODE_GLSL_HERO`: Hardware GLSL Shader Mode. Enables full-screen shader execution beneath standard SDL2 drawing.
 - *[CRITICAL] Clear Color Rule in GLSL_HERO:* You MUST call `tincle_clear((TinCLE_Color){0, 0, 0, 0})` with **Alpha = 0**. Specifying an opaque alpha will erase/hide the shader layer beneath SDL rendering.
- **Inputs (Strict Pure SDL2 Constants):** No custom engine macros exist. Use native SDL2 constants directly:
- *Mouse:* `SDL_BUTTON_LEFT`, `SDL_BUTTON_RIGHT`, etc. (via `tincle_mouse_pressed`)
- *Gamepad:* `SDL_CONTROLLER_BUTTON_A`, `SDL_CONTROLLER_BUTTON_DPAD_UP`, etc. (via `tincle_pad_pressed`)
- *Keyboard:* `SDL_SCANCODE_A`, `SDL_SCANCODE_SPACE`, etc. (via `tincle_key_pressed`)
- **Color Structure (`TinCLE_Color`):** Identical/compatible with `SDL_Color`.

```
typedef struct { unsigned char r, g, b, a; } TinCLE_Color;
```

- *No Color Macros:* Create colors inline using standard C99 compound literals. Example: `(TinCLE_Color){255, 0, 0, 255}`.
- **[CRITICAL ANTI-BUG] TCC Compound Literal Restriction:** TCC cannot parse compound literals in global/static array initializers. Declare arrays globally, but assign values sequentially inside `game_load()`.
 - *Forbidden:* `static TinCLE_Color c[] = { (TinCLE_Color){255,0,0,255} };`
 - *Mandatory Remedy:* `static TinCLE_Color c[1]; void game_load() { c[0] = (TinCLE_Color){255,0,0,255}; }`

5. API Reference & Lifecycle

- **Canvas:** Default virtual resolution is `640 x 480`.
- **Rendering Rules:** `tincle_clear` must be at the start of drawing. **IMAGE COORDINATES ARE CENTER-BASED (NOT TOP-LEFT).**
- **State Rule:** `tincle_change_state` must be called at the very end of `game_load()` to avoid a black screen.

Function Signature	Description & Critical Notes [Args / Return]
<pre>void tincle_start(TinCLE_InitCallback init_cb, const char* title, bool is_debug, int w, int h, TinCLE_RenderMode mode);</pre>	Main Entry. Bootstraps loop. • <code>is_debug</code> : <code>true</code> spawns Win debug console (<code>printf</code>); <code>false</code> hides it for release. • <code>mode</code> : Set to <code>TINCLE_MODE_2D_NORMAL</code> or <code>TINCLE_MODE_GLSL_HERO</code>.
<pre>void tincle_add_state(int state_id, TinCLE_UpdateCallback update_cb, TinCLE_DrawCallback draw_cb);</pre>	Binds state ID to standard callbacks: <code>update(float dt)</code> and <code>draw(void)</code> .
<pre>void tincle_change_state(int state_id);</pre>	[CRITICAL] Switches state. Call at end of <code>game_load()</code> or viewport stays black.
<pre>int tincle_key_pressed(int scancode);</pre>	Returns <code>1</code> if key held. Must use pure <code>SDL_SCANCODE_*</code> .
<pre>void tincle_get_mouse(int* x, int* y);</pre>	Outputs current mouse (x, y) dynamically scaled to virtual resolution.
<pre>int tincle_mouse_pressed(int button_type);</pre>	Returns <code>1</code> if mouse held. Must use pure <code>SDL_BUTTON_*</code> (e.g. <code>LEFT</code>).
<pre>int tincle_pad_pressed(int pad_button);</pre>	Returns <code>1</code> if pad button held. Must use pure <code>SDL_CONTROLLER_BUTTON_*</code> .

Function Signature	Description & Critical Notes [Args / Return]
<code>float tincle_pad_axis_x(void);</code>	Left stick X (-1.0f to 1.0f). Has built-in <5000 hardware deadzone filter.
<code>float tincle_pad_axis_y(void);</code>	Left stick Y (-1.0f to 1.0f). Has built-in <5000 hardware deadzone filter.
<code>void tincle_clear(TinCLE_Color color);</code>	[MANDATORY] Clears screen. Call at absolute start of every <code>draw()</code> callback. Alpha must be 0 in GLSL_HERO mode.
<code>int tincle_load_font(const char* filename, int size);</code>	Loads TTF at pixel size. Returns unique Font ID (int).
<code>void tincle_set_font(int font_id);</code>	Sets active global font by Font ID for subsequent text calls.
<code>void tincle_draw_text(const char* text, int x, int y, TinCLE_Color color);</code>	Draws UTF-8 text. Anchor: Top-Left (x, y). Uses active font.
<code>void tincle_draw_text_center(const char* text, int x, int y, TinCLE_Color color);</code>	Draws UTF-8 text. Anchor: EXACT CENTER (x, y). Uses active font.
<code>int tincle_load_image(const char* filename);</code>	Caches image (PNG/JPG). Returns Image ID (<code>img_id</code>), or <code>-1</code> .
<code>void tincle_draw_image(int img_id, int x, int y);</code>	Uses <code>img_id</code> . [CRITICAL GEAR] Anchor: EXACT CENTER (x, y) of unscaled image.
<code>void tincle_draw_image_ex(int img_id, int x, int y, int w, int h, double angle);</code>	Uses <code>img_id</code> . [CRITICAL GEAR] Anchor: EXACT CENTER (x, y) for position, scale, and rotation.
<code>void tincle_play_music(const char* filename);</code>	Plays Ogg BGM immediately on infinite loop. Unloads previous music automatically.

Function Signature	Description & Critical Notes [Args / Return]
<code>void tinkle_stop_music(void);</code>	Instantly stops and unloads current BGM from memory.
<code>int tinkle_load_sound(const char* filename);</code>	Caches WAV/Ogg SE clip. Returns unique Sound ID, or -1 .
<code>void tinkle_play_sound(int se_id);</code>	Plays SE channel via auto-mixer (supports up to 16 concurrent sounds).
<code>struct SDL_Renderer* tinkle_get_renderer(void);</code>	Returns native `SDL_Renderer*` for direct SDL2/SDL2_gfx primitive drawing.
<code>const char* tinkle_get_version(void);</code>	Returns read-only string of engine version (e.g., <code>"0.05"</code>).
<code>bool tinkle_is_fullscreen(void);</code>	Returns <code>true</code> if current window is in full-screen mode.
<code>void tinkle_set_cursor_auto_policy(bool visible_in_windowed, bool visible_in_fullscreen);</code>	Sets default cursor visibility behavior for Windowed and Fullscreen modes.
<code>void tinkle_set_cursor_visible(bool visible);</code>	Toggles engine software cursor visibility instantly.
<code>void tinkle_set_custom_cursor(int img_id, int hotspot_x, int hotspot_y);</code>	Replaces software cursor with image ID and hotspot offsets.
<code>void tinkle_reset_cursor_to_default(void);</code>	Restores default mouse cursor image (<code>cursor.png</code>).

5.1 GLSL Unleashed API (`TINCLE_MODE_GLSL_HERO` Only)

- [CRITICAL Shader Code Rules]:**

- Shaders **MUST** be 100% GLES2 compliant to run on WebAssembly.
- NEVER write** `precision mediump float;` **in your** `.glsl` **file!** TinCLE automatically injects this header during the Emscripten build. Writing it manually

causes syntax errors on Windows OpenGL.

3. [MANDATORY] Viewport & Aspect Ratio Correction for Fullscreen /

Letterboxing:

- **NEVER** calculate UV coordinates using `gl_FragCoord.xy / u_resolution` directly. When entering fullscreen mode or letterboxing, `u_resolution` represents the physical window/screen dimensions, causing critical position shifts.
- **ALWAYS** calculate UV coordinates using `u_viewport` to strictly lock the rendering area to the actual game viewport: `vec2 uv = (gl_FragCoord.xy - u_viewport.xy) / u_viewport.zw;`
- When normalizing mouse coordinates received from C (`u_custom_mouse.xy`), apply a fixed aspect ratio matching the virtual game resolution (e.g., `640.0 / 480.0`) to both `p.x` and `m.x` to prevent horizontal distortion and mouse offset across different screen sizes.

• Automatic Shader Uniforms (Built-in):

- `uniform float u_time;` -> In-game time in seconds (`SDL_GetTicks() / 1000.0f`).
- `uniform vec2 u_resolution;` -> Physical window/screen resolution (`width`, `height`).
- `uniform vec4 u_viewport;` -> Actual rendering viewport rect (`x`, `y`, `w`, `h`).

Function Signature	Description & Critical Notes
<code>void tingle_render_shader(const char* filename);</code>	Call inside <code>game_load()</code> to keep a background shader running continuously.
<code>int tingle_load_shader(const char* filename);</code>	Call inside <code>game_load()</code> . Compiles shader and returns Shader ID (<code>int</code>).
<code>void tingle_set_shader(int shader_id);</code>	Sets active background shader ID (<code>-1</code> to disable).
<code>void tingle_shader_use(int shader_id);</code>	Activates shader for uniform updates. Has overhead (<code>glUseProgram</code>). Call only when switching scenes or updating values.

Function Signature	Description & Critical Notes
<code>void tincle_shader_send_vec3(int shader_id, const char* name, float x, float y, float z);</code>	Binds a <code>vec3</code> uniform value to the target shader.
<code>void tincle_shader_send_float(int shader_id, const char* name, float value);</code>	Binds a <code>float</code> uniform value to the target shader.

6. Mandatory AI Coding Rules & Template

[CRITICAL WARNING] Never output search citations or footnotes (e.g., `[cite: 1]`, `[1]`) anywhere in or around code blocks. They instantly cause fatal C syntax errors. Output ONLY valid, clean C source code. The AI MUST strictly comply 100% with this structural template and the absolute rules below:

```
#include "libs/tincle.h" // MANDATORY: Always include engine core first
#include <math.h>         // Explicitly include if math functions (sinf, etc.) are used
#ifdef __EMSCRIPTEN__
    #include <SDL2/SDL_scancode.h> // Example: Web (Use angle brackets <>)
    #include <SDL2/SDL2_gfxPrimitives.h> // For native SDL2_gfx on Web
#else
    #include "libs/SDL_scancode.h" // Example: Windows (Use Local double quotes "")
    #include "libs/SDL2_gfxPrimitives.h" // For native SDL2_gfx on Windows
    #include <windows.h> // Required for native Windows execution
#endif
// --- State Callbacks Example ---
void title_update(float dt) {} // RULE: ALL movements/timers must multiply by 'dt'
void title_draw(void) {
    // CRITICAL: Use Alpha = 0 when in GLSL_HERO mode to keep shader visible!
    tincle_clear((TinCLE_Color){0, 0, 0, 0});
}
void game_load(void) {
    // RULE: MUST execute load_font as the first line here. Assign to int handle.
    int f = tincle_load_font("font.ttf", 32);
    // Example GLSL Setup (If using TINCLE_MODE_GLSL_HERO)
    // int shader_id = tincle_load_shader("bg.glsl");
    // tincle_render_shader("bg.glsl");
    // --- Example: Correct Resource Loading (Load ONLY ONCE here!) ---
    // g_custom_cursor = tincle_load_image("my_cursor.png");
```

```

// tincle_set_custom_cursor(g_custom_cursor, 0, 0);
// RULE: MUST register states exclusively inside game_load()
tincle_add_state(0, title_update, title_draw);
tincle_change_state(0);
}
#ifdef __EMSCRIPTEN__
int main(int argc, char* argv[]) { tincle_start(game_load, "TITLE", true, 640,
480, TINCLE_MODE_2D_NORMAL); return 0; }
#else
int WINAPI WinMain(HINSTANCE h, HINSTANCE hp, LPSTR c, int n) { tincle_start(ga
me_load, "TITLE", true, 640, 480, TINCLE_MODE_2D_NORMAL); return 0; }
#endif

```

7 Absolute Rules for Code Generation:

1. **Pure C99 Standard:** Absolute C syntax only. Never use C++ styles (classes, namespaces).
2. **Delta Time (dt):** All locomotion, physics, and custom timers must be multiplied by `dt` in `update()`.
3. **Input Synthesis:** Overwrite gamepad analog vector (`tincle_pad_axis`) with max keyboard constants (`1.0f` / `-1.0f`) if arrows/WASD are pressed.
4. **Header Scopes & SDL2 Usage:** For extra SDL2/SDL2_gfx headers, use `<SDL2/filename.h>` inside `__EMSCRIPTEN__` and `"libs/filename.h"` inside `#else` (like `SDL2_gfxPrimitives.h`). You can freely use `tincle_get_renderer()` for custom SDL2/SDL2_gfx drawing extensions, but NEVER modify window dimensions or destroy/alter core SDL window/renderer loops directly.
5. **Shader Normalized Mouse Coordinates (GLSL_HERO Mode):**
 - Mouse coordinates returned by `tincle_get_mouse()` are automatically scaled to the virtual resolution (e.g., 640x480).
 - When passing mouse positions to custom GLSL shaders, C code **MUST** normalize values strictly using the virtual game resolution and invert the Y-axis for OpenGL space: `s_mouse_nx = (float)mx / VIRTUAL_WIDTH;` `s_mouse_ny = 1.0f - ((float)my / VIRTUAL_HEIGHT);`
 - Do NOT pass raw screen pixel coordinates directly to the shader if `u_viewport` is used for screen-space UV calculation.
6. Ensure all shader files (`.glsl`) are placed in the root folder and listed in `exsass.txt`.
7. **Strict State Model:** Split logic cleanly via State Management API (e.g., Title, Play, Over). Never write a raw single-loop frame block.
8. **Toolchain Distribution Policy:** Instruct users to use `tinclec` for Win EXE and `emcass` for Web builds. `asspack` handles media asset bundling (supports auto-

scanning via `exsass.txt` when called without arguments); `icondom` generates ICO files; `poopee` handles metadata embedding into compiled EXEs.

7. Production Release Pipelines (Distribution Rules)

When generating final release builds or instructions, the AI **MUST** follow this 4-step pipeline and enforce documentation mandates.

Step 1: Prep Assets & Hide Console

- Update `CREDITS.txt` with third-party asset attributions. Ensure it is in the root directory.
- Ensure `cursor.png` and `exsass.txt` are configured in the project root.
- Set the 3rd argument (`is_debug`) of `tincle_start` to `false` in `game.c` to hide the command console.

Step 2: Windows Desktop Build Path (ZIP Pack)

1. **Compile:** `tinclec game.exe game.c [other_sources.c...]`
2. **Generate Icon:** If a multi-size `.ico` is not yet prepared, convert an image via `icondom.exe` :

```
icondom icon.png icon.ico
```

3. **Embed Resources (Optional):** Embed the generated icon and Windows manifest:

```
poopee game.exe icon.ico manifest.xml
```

(Use `"none"` to skip ico/manifest).

4. **Pack Assets:** Pack files for final distribution:
 - **Auto Pack (Recommended):** Simply run `asspack` with no arguments to automatically collect all assets/shaders matching `exsass.txt` :

```
asspack
```

- **Manual Pack:** Alternatively, specify files explicitly: `asspack font.ttf cursor.png bgm.ogg [images/shaders/sounds...]`
 - (Creates `game.data` . Do not ship raw assets).*
5. **ZIP Checklist (8 Items Only):** `game.exe` , `game.data` , `tincle.dll` , `SDL2.dll` , `SDL2_ttf.dll` , `SDL2_gfx.dll` , `CREDITS.txt` , `readme.txt` .

Step 3: Web Browser Build Path (ZIP Pack)

1. **Verify Asset Extensions:** Ensure all asset AND shader extensions are listed in `exsass.txt`.
2. **Compile & Pack:** `emccass index.html "Game Title" [Width] [Height] game.c [other_sources.c...]`
 - **[CRITICAL] Always output exactly as `index.html`**. Name modification is strictly forbidden for itch.io/web platforms. Automatically outputs `index.js`, `index.wasm`, and `index.data`.
 - [Width] and [Height] refer to the actual dimensions of the game screen in the web version. Specify values that match the game screen dimensions.
3. **ZIP Checklist (5 Items Only):** `index.html`, `index.js`, `index.wasm`, `index.data`, `CREDITS.txt`.
4. **[TROUBLESHOOTING] Server Header Requirements (.htaccess):** If the Web build fails to run and throws the following browser console error: `Uncaught (in promise) TypeError: Failed to execute 'decode' on 'TextDecoder': The provided ArrayBuffer value must not be resizable` This indicates that critical Emscripten capabilities (such as SharedArrayBuffer) are blocked due to missing Cross-Origin Isolation headers on the web host.
 - **Remedy:** Instruct the user to place the bundled `.htaccess` file into the same directory as `index.html` to enable the required COOP and COEP security headers on Apache-based servers:

```
<IfModule mod_headers.c>
  Header add Cross-Origin-Opener-Policy "same-origin"
  Header add Cross-Origin-Embedder-Policy "require-corp"
</IfModule>
```

Step 4: Mandatory `readme.txt` Specifications

AI must explicitly include these details in `readme.txt` or platform descriptions:

- **Controls Disclosure:** State that `Alt + Enter` toggles Fullscreen, and `Alt + Backspace` resets window scale on Windows. State that the canvas button toggles Fullscreen on Web.
- **Creator Copyright:** Display user's own copyright notice (`Copyright (C) 2026 [Developer Name]`).

8. Complete Boilerplate Code (`game.c` Template)

The AI can utilize this robust, production-ready full skeleton code to initiate or rewrite any game project under TinCLE v0.05.

```
#define SDL_MAIN_HANDLED
#include "libs/tincle.h"
#include <stdio.h>
#include <math.h>
#ifdef __EMSCRIPTEN__
    #include <SDL2/SDL_scancode.h>
    #include <SDL2/SDL2_gfxPrimitives.h>
#else
    #include "libs/SDL_scancode.h"
    #include "libs/SDL2_gfxPrimitives.h"
    #include <windows.h>
#endif
// [TCC WORKAROUND] Prevent macro collision and fix missing float math symbols
#undef M_PI
#define M_PI 3.14159265358979323846
#ifndef sinf
#define sinf(x) ((float)sin((double)(x)))
#endif
// --- Globals & Assets ---
static SDL_Renderer* renderer = NULL;
static float g_player_x = 320.0f;
static float g_player_y = 240.0f;
static float g_game_time = 0.0f;
static int g_img_player = -1;
static int g_font_large = -1;
static int g_font_medium = -1;
// --- State 0: Title ---
void title_update(float dt) {
    if (tincle_key_pressed(SDL_SCANCODE_SPACE) || tincle_pad_pressed(SDL_CONTROLLER_BUTTON_START)) {
        tincle_change_state(1); // Play State
    }
}
void title_draw(void) {
    tincle_clear((TinCLE_Color){0, 0, 0, 255});
    tincle_set_font(g_font_large);
    tincle_draw_text_center("YOUR_GAME_TITLE", 320, 180, (TinCLE_Color){255, 255, 255, 255});
    tincle_set_font(g_font_medium);
    tincle_draw_text_center("PRESS SPACE TO START", 320, 300, (TinCLE_Color){0, 255, 0, 255});
}
// --- State 1: Gameplay ---
void play_update(float dt) {
```

```

    g_game_time += dt;
    float move_speed = 200.0f; // Multiplied by dt for framerate independence
    if (tincle_key_pressed(SDL_SCANCODE_LEFT) || tincle_pad_pressed(SDL_CONTROLLER_BUTTON_DPAD_LEFT)) g_player_x -= move_speed * dt;
    if (tincle_key_pressed(SDL_SCANCODE_RIGHT) || tincle_pad_pressed(SDL_CONTROLLER_BUTTON_DPAD_RIGHT)) g_player_x += move_speed * dt;
    if (tincle_key_pressed(SDL_SCANCODE_UP) || tincle_pad_pressed(SDL_CONTROLLER_BUTTON_DPAD_UP)) g_player_y -= move_speed * dt;
    if (tincle_key_pressed(SDL_SCANCODE_DOWN) || tincle_pad_pressed(SDL_CONTROLLER_BUTTON_DPAD_DOWN)) g_player_y += move_speed * dt;
    if (tincle_key_pressed(SDL_SCANCODE_ESCAPE) || tincle_pad_pressed(SDL_CONTROLLER_BUTTON_BACK)) {
        tincle_change_state(0); // Title State
    }
}

void play_draw(void) {
    tincle_clear((TinCLE_Color){20, 20, 50, 255});
    tincle_set_font(g_font_medium);
    tincle_draw_text("GAME PLAYING", 20, 20, (TinCLE_Color){0, 255, 0, 255});
    // CRITICAL: tincle_draw_image handles coordinates as the EXACT CENTER anchor point
    if (g_img_player != -1) {
        tincle_draw_image(g_img_player, (int)g_player_x, (int)g_player_y);
    } else {
        tincle_draw_text("P", (int)g_player_x, (int)g_player_y, (TinCLE_Color){255, 0, 0, 255});
    }
    // Native API Extension Examples (Direct renderer usage)
    SDL_Rect rect = { 100, 100, 50, 50 };
    SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);
    SDL_RenderFillRect(renderer, &rect);
    filledEllipseRGBA(renderer, 400, 300, 50, 25, 255, 0, 0, 128);
    int tri_top_y = 150 + (int)(sinf(g_game_time * 5.0f) * 15.0f);
    filledTrigonRGBA(renderer, 400, tri_top_y, 360, 200, 440, 200, 0, 255, 255, 200);
}

// --- Initialization & Callbacks ---
void game_load(void) {
    renderer = tincle_get_renderer(); // Cache renderer context for native loops
    g_font_large = tincle_load_font("font.ttf", 64);
    g_font_medium = tincle_load_font("font.ttf", 32);
    g_img_player = tincle_load_image("player.png");
    tincle_add_state(0, title_update, title_draw);
    tincle_add_state(1, play_update, play_draw);
    tincle_change_state(0); // Activate initial state
}

// --- Entry Point Bootstrapper ---
#ifdef __EMSCRIPTEN__

```

```
int main(int argc, char* argv[]) { tinkle_start(game_load, "YOUR_GAME_TITLE", true, 640, 480, TINCLE_MODE_2D_NORMAL); return 0; }  
#else  
int WINAPI WinMain(HINSTANCE h, HINSTANCE hp, LPSTR c, int n) { tinkle_start(game_load, "YOUR_GAME_TITLE", true, 640, 480, TINCLE_MODE_2D_NORMAL); return 0; }  
#endif
```