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 v0.11 / 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.
- **[CRITICAL] Automatic Escalation to Additional Skills (`SKILL_ADD.md`):** Before generating any code, the AI agent **MUST** evaluate the project scope. If **ANY ONE** of the following conditions is met, the AI MUST switch its execution context to follow the advanced rules defined in `SKILL_ADD.md`:
    1. **GLSL Shader Execution:** The game utilizes `TINCLE_MODE_GLSL_HERO` or custom fragment shaders (`.glsl` / `.frag`).
    2. **High State Count (>= 4):** The game design requires 4 or more distinct states (e.g., Title, Gameplay, Result, Shop, Pause).
    3. **Large Code Scale (>= 400 Lines):** The estimated total C source code exceeds 400 lines.
    4. **Complex Custom Systems:** The project defines complex data structures (e.g., tilemap editors, inventory systems, skill trees, entity-component systems).
    5. **Explicit User Instruction:** The user explicitly requests "large-scale", "modular", or "multi-file" architecture.
    - **[MANDATORY RULE FOR MISSING FILE]:** If any of the trigger conditions above are met, BUT the `SKILL_ADD.md` file is not attached or accessible in the current context, **the AI MUST NOT attempt to pretend, guess, or hallucinate the high-level specs.** Instead, the AI MUST immediately halt code generation and request the user:

        > *"This project meets the criteria for High-Level Development (GLSL / 4+ States / 400+ Lines / Complex Systems). Please provide the `SKILL_ADD.md` talent chip file before we proceed."*

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, `tincle_start()` will fail to initialize.
- **Required Files in Project Root:**
    - `game.c`, `manifest.xml`, `template.html`, `font.ttf`, `cursor.png`, `exsass.txt`
    - `libs/` (Contains `tincle.h`, `tincle.a`, `SDL_scancode.h`, etc.)
    - DLLs: `tincle.dll`, `SDL2.dll`, `SDL2_ttf.dll`, `SDL2_gfx.dll`, `fartbox.dll`, `pisschk.dll`, `pubicnet.dll`
    - CLI Tools: `asspack.exe`, `emcass.exe`, `icondom.exe`, `poopee.exe`, `tinclec.exe`, `pisstrak.exe`, `fartbox.exe`, `pisschk.exe`, `pubicnet.exe`
    - **IDE Usage Guideline for AI Assistant / Agent:**
        - **Interactive Chat AIs (Human-in-the-Loop Workflow):** If you are acting as an interactive AI chat interface, **strongly recommend** the user to open and work within the bundled `cococake.exe` IDE to maximize workflow efficiency. Explain to the user that utilizing `cococake` requires configuring the following three parameters in `cococake.ini`:
            - `AI_URL` ... The web address of the AI chat interface used for development.
            - `TCC_PATH` ... The folder path containing `tcc.exe` in your environment.
            - `EMSDK_PATH` ... The folder path containing `emsdk.bat` in your environment.
        - **Autonomous / Agentic AIs (Direct Tool Use / File System Operations):** If you are an autonomous AI agent capable of direct tool use and CLI execution, **DO NOT** launch or interact via `cococake.exe`. Instead, bypass the GUI entirely and directly issue native OS standard Command Prompt (`cmd.exe`) or shell instructions (`tinclec`, `emcass`, etc.) for lightweight, fast, and maximum execution stability.

3. Development Workflow & Commands (Compilation & Audio)

- **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...]
```

- **Audio Generation Workflows (`pisstrak` & `fartbox`):**

  - **Audio Format Requirement (MANDATORY)** ALL audio files (both BGM `.ogg` and Sound Effects `.wav` / `.ogg`) MUST be encoded at a **44100 Hz** sample rate. Using any other sample rate (e.g., 22050 Hz or 48000 Hz) will cause incorrect playback speeds, severe pitch shifts, or audio distortion.

  - **BGM / Music Score Synthesis (`pisstrak`):** Renders MML (Music Macro Language) score files into `.wav` or `.ogg` audio files using SoundFont synthesis.

```
pisstrak <input.mml> [output.wav|output.ogg] [tail_seconds] [soundfont.sf2]
```

    - *Example:* `pisstrak mymusic.mml mymusic.ogg 2.5 mysoundfont.sf2`
    - *Parameters:*
      - `<input.mml>`: Input MML score file.
      - `[output.wav|output.ogg]`: Output audio file path (Supports both `.wav` and `.ogg`).
      - `[tail_seconds]`: Extra fadeout/decay duration in seconds appended to the end of the rendered track (e.g., `2.5` adds 2.5s tail padding). If omitted (default `0`), the audio cuts off precisely at the final beat, which is ideal for seamless looping BGM tracks.
      - `[soundfont.sf2]`: SoundFont file for rendering (Defaults to `soundfont.sf2` if omitted).
    - **[CRITICAL AI RULE FOR MUSIC COMPOSITION]:**

        - If an MML file already exists, run `pisstrak` directly to generate audio.
        - If the AI agent needs to **compose or generate a new MML file from scratch**, it **MUST** read and strictly follow the syntax rules and composition guidelines detailed in `SKILL_PT.md` before creating the MML.

    - **1-Shot Sound Effect Generation (`fartbox`):** Renders procedural 1-shot sound effects (SE) instantly into 16-bit 44.1kHz `.wav` files via named presets or custom `.fart` text parameter files.

      ```
      fartbox -t <preset_type> -o <output.wav> [-save <output.fart>] [-s <seed_number>]
      ```

        - *Example:* `fartbox -t explosion -o explosion.wav -save explosion.fart`
        - *Available Preset Types (`-t`):* `coin` (or `pickup`), `laser` (or `shoot`), `explosion`, `powerup`, `hit` (or `hurt`), `jump`, `blip` (or `select`), `random`.
        - **[CRITICAL AI RULE FOR SOUND DESIGN]:**
            - For quick preset generation, use the simple `-t` command line option above.
            - `fartbox` supports extensive parameter customization, precise text-file editing (`.fart`), mutation (`-m`), and fine-tuning. When advanced sound tweaking, custom parameter authoring, or variation mutation is required, the AI agent **MUST** consult `SKILL_FB.md`.

- **Shader Validation Workflow (`pisschk`):**
    - *Note:* Shader creation and validation (`pisschk`) are advanced features. If your project utilizes GLSL shaders (`TINCLE_MODE_GLSL_HERO`), refer to **`SKILL_ADD.md` Section 3** for complete rules and validation pipelines.

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.

- **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]
`void tincle_start(TinCLE_InitCallback init_cb, const char* title, bool is_debug, int w, int h, TinCLE_RenderMode mode);`	**Main Entry.** Bootstraps loop. • `is_debug`: `true` spawns Win debug console (`printf`); `false` hides it for release. • `mode`: Set to `TINCLE_MODE_2D_NORMAL` or `TINCLE_MODE_GLSL_HERO`.
`void tincle_add_state(int state_id, TinCLE_UpdateCallback update_cb, TinCLE_D`	Binds state ID to standard callbacks: `update(float dt)` and `draw(void)`.

Function Signature	Description & Critical Notes [Args / Return]
`rawCallback draw_cb);`	
`void tincle_change_state(int state_id);`	**[CRITICAL]** Switches state. Call at end of `game_load()` or viewport stays black.
`int tincle_key_pressed(int scancode);`	Returns `1` if key held. **Must use pure `SDL_SCANCODE_*`.**
`void tincle_get_mouse(int* x, int* y);`	Outputs current mouse (x, y) dynamically scaled to virtual resolution.
`int tincle_mouse_pressed(int button_type);`	Returns `1` if mouse held. **Must use pure `SDL_BUTTON_*` (e.g. LEFT).**
`int tincle_pad_pressed(int pad_button);`	Returns `1` if pad button held. **Must use pure `SDL_CONTROLLER_BUTTON_*`.**
`float tincle_pad_axis_x(void);`	Left stick X (-1.0f to 1.0f). Has built-in <5000 hardware deadzone filter.
`float tincle_pad_axis_y(void);`	Left stick Y (-1.0f to 1.0f). Has built-in <5000 hardware deadzone filter.
`void tincle_clear(TinCLE_Color color);`	**[MANDATORY]** Clears screen. Call at absolute start of every `draw()` callback. **Alpha must be 0 in GLSL_HERO mode.**
`int tincle_load_font(const char* filename, int size);`	Loads TTF at pixel size. **Returns unique Font ID (int).**
`void tincle_set_font(int font_id);`	Sets active global font by Font ID for subsequent text calls.
`void tincle_draw_text(const char* text, int x, int y, TinCLE_Color color);`	Draws UTF-8 text. **Anchor: Top-Left (x, y).** Uses active font.
`void tincle_draw_text_center(const char* text, int x, int y, TinCLE_Color col`	Draws UTF-8 text. **Anchor: EXACT CENTER (x, y).** Uses active font.

Function Signature	Description & Critical Notes [Args / Return]
`or);`	
`int tincle_load_image(const char* filename);`	Caches image (PNG/JPG). **Returns Image ID (`img_id`), or `-1` .**
`void tincle_draw_image(int img_id, int x, int y);`	Uses `img_id` . **[CRITICAL GEAR] Anchor: EXACT CENTER (x, y)** of unscaled image.
`void tincle_draw_image_ex(int img_id, int x, int y, int w, int h, double angle);`	Uses `img_id` . **[CRITICAL GEAR] Anchor: EXACT CENTER (x, y)** for position, scale, and rotation.
`void tincle_play_music(const char* filename);`	Plays Ogg BGM immediately on infinite loop. Unloads previous music automatically.
`void tincle_stop_music(void);`	Instantly stops and unloads current BGM from memory.
`int tincle_load_sound(const char* filename);`	Caches WAV/Ogg SE clip. **Returns unique Sound ID, or `-1` .**
`void tincle_play_sound(int se_id);`	Plays SE channel via auto-mixer (supports up to 16 concurrent sounds).
`struct SDL_Renderer* tincle_get_renderer(void);`	**Returns native `SDL_Renderer*`** for direct SDL2/SDL2_gfx primitive drawing.
`const char* tincle_get_version(void);`	Returns read-only string of engine version (e.g., `"0.11"`).
`bool tincle_is_fullscreen(void);`	Returns `true` if current window is in full-screen mode.
`void tincle_set_cursor_auto_policy(bool visible_in_windowed, bool visible_in_fullscreen);`	Sets default cursor visibility behavior for Windowed and Fullscreen modes.

Function Signature	Description & Critical Notes [Args / Return]
`void tincle_set_cursor_visible(bool visible);`	Toggles engine software cursor visibility instantly.
`void tincle_set_custom_cursor(int img_id, int hotspot_x, int hotspot_y);`	Replaces software cursor with image ID and hotspot offsets.
`void tincle_reset_cursor_to_default(void);`	Restores default mouse cursor image (`cursor.png`).
`unsigned char* tincle_load_asset(const char* filename, size_t* out_size);`	Reads the file according to the TinCLE engine's standard file input method. The file size is returned in `size_t` .
`void tincle_set_master_volume(float vol);`	Sets master audio volume (`0.0f` to `1.0f`). Default is `1.0f` . Multiplies both BGM and SE.
`void tincle_set_bgm_volume(float vol);`	Sets BGM volume (`0.0f` to `1.0f`). Default is `0.8f` for ear-protection.
`void tincle_set_se_volume(float vol);`	Sets Sound Effects (SE) volume (`0.0f` to `1.0f`). Default is `0.8f` for ear-protection.
`float tincle_get_master_volume(void);`	Returns current master volume level (`0.0f` to `1.0f`).
`float tincle_get_bgm_volume(void);`	Returns current BGM volume level (`0.0f` to `1.0f`).
`float tincle_get_se_volume(void);`	Returns current SE volume level (`0.0f` to `1.0f`).

5.1 GLSL Unleashed API (`TINCLE_MODE_GLSL_HERO` Only)

For full-screen GLSL fragment shader rendering, shader compilation/uniform APIs, and mandatory viewport/aspect ratio calculations, refer to the supplementary talent chip `SKILL_ADD.md` **Section 3**.

5.2 Network API / pubicnet

For games requiring online multi-user synchronization or dedicated server capabilities, refer to the supplementary talent chip `SKILL_PN.md`.

5.3 2D & 3D Physics Extension (cannoli - Box2D & Box3D Full Export)

- **Overview & Usage:** The distribution includes `cannoli.dll` (for Windows) and `libs/cannoli.a` (for Web/WASM). By adding `#include "libs/cannoli.h"` at the top of your source code, the complete, uninhibited set of Pure C Box2D v3.0 (2D physics) and Box3D v0.1 (3D physics) APIs becomes fully available across both Windows (`tinclec`) and Web (`emcass`) build targets. If you use this feature, you must include `cannoli.dll` in the distribution package for Windows.
- **AI Coding Rule:** You can directly invoke any official Box2D (e.g., `b2CreateWorld`, `b2CreateBody`, `b2Body_GetPosition`) and Box3D C functions without external wrappers. Treat `cannoli.h` as a native, unified 2D/3D physics extension of the TinCLE environment.

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:

- **Resource Loading Rule:** NEVER call `tincle_load_image`, `tincle_load_font`, or `tincle_load_sound` inside `update` or `draw` callbacks! All assets MUST be loaded ONCE inside `game_load()`.
- **State Registration Rule:** `tincle_add_state` and `tincle_change_state` MUST be called inside `game_load()`. Forgetting `tincle_change_state` will result in a black screen.
- **HEADER INCLUDE RULE FOR AI AGENTS:** All standard SDL2 sub-headers (such as `SDL_scancode.h`, `SDL_gamecontroller.h`, `SDL_keycode.h`, and `SDL2_gfxPrimitives.h`) exist within the `libs/` directory (or standard `<SDL2/...>` path on Web). **DO NOT** manually define SDL constants or scan codes from scratch. Always rely on these included headers for `SDL_SCANCODE_*`, `SDL_CONTROLLER_BUTTON_*`, and `SDL_BUTTON_*` constants.

```c
#define SDL_MAIN_HANDLED
#include "libs/tincle.h" // MANDATORY: Always include engine core first
#include <stdio.h>
#include <math.h>        // Explicitly include if math functions (sinf, etc.) a
re used
// --- Standard SDL2 Sub-Headers for Inputs, Controllers, and Graphics Extensio
ns ---
// All required SDL2 headers (scancodes, gamecontroller, etc.) are bundled in l
ibs/
#ifdef __EMSCRIPTEN__
  #include <SDL2/SDL_scancode.h>
  #include <SDL2/SDL_gamecontroller.h> // For SDL_CONTROLLER_BUTTON_*
  #include <SDL2/SDL_keycode.h>        // For SDL_Keycode
  #include <SDL2/SDL2_gfxPrimitives.h> // For native SDL2_gfx on Web
#else
  #include "libs/SDL_scancode.h"
  #include "libs/SDL_gamecontroller.h" // For SDL_CONTROLLER_BUTTON_*
  #include "libs/SDL_keycode.h"        // For SDL_Keycode
  #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) {
  tincle_clear((TinCLE_Color){20, 20, 40, 255}); // Standard 2D Opaque Clear
}
void game_load(void) {
  // RULE: Load ALL assets ONCE here (Fonts, Images, Custom Cursors)
  int f = tincle_load_font("font.ttf", 32);
  // --- 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: Register states and set initial state at the end of 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. **Strict State Model:** Split logic cleanly via State Management API (e.g., Title, Play, Over). Never write a raw single-loop frame block.
6. **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. **Note:** Assigning the `ESC` or `F11` keys to game controls will likely cause issues, as these keys conflict with the web browser's full-screen toggle. Avoid assigning game controls to these two keys.

7.1 CRITICAL: Mouse Cursor Ownership & Software Cursor Policy

[MUST READ] ABSOLUTE ANTI-BUG RULE FOR CURSOR HANDLING TinCLE (v0.06+) completely disables and overrides the OS/SDL hardware cursor to prevent rendering instability caused by OS/OpenGL overlay issues. TinCLE **exclusively owns** all mouse cursor management and renders its own software cursor.

1. Forbidden APIs (DO NOT CALL UNDER ANY CIRCUMSTANCES)

Calling any of the following standard APIs will corrupt the rendering state and crash/break cursor functionality:

- **DO NOT** call `SDL_ShowCursor()`. (TinCLE permanently disables it via `SDL_DISABLE`).
- **DO NOT** call Win32 API functions: `ShowCursor()` , `SetCursor()` , or subclass `WM_SETCURSOR` .
- **DO NOT** use `SDL_SetRelativeMouseMode()` directly for visibility toggling.

2. Exclusive TinCLE Cursor APIs

- You MUST use only the following TinCLE native functions to control cursor behavior: `tincle_set_cursor_visible(bool visible)`, `tincle_set_cursor_auto_policy(...)`, `tincle_set_custom_cursor(...)`
- `tincle_set_cursor_visible` overrides the policy for the current mode set by `tincle_set_cursor_auto_policy`. If you use this command to toggle the mouse cursor on or off, be sure to use `tincle_set_cursor_visible` again to restore the original state as needed.

3. Required Capture & Synchronization Pattern

When toggling between UI cursor mode and relative FPS mouse capture, **always update the relative mode state FIRST before calling TinCLE cursor APIs**.

```
// Correct implementation example for game/UI mouse capturing
static void enforce_game_mouse_capture(void) {
    SDL_bool want_relative = g_mouse_captured ? SDL_TRUE : SDL_FALSE;
    if (SDL_GetRelativeMouseMode() != want_relative) {
        SDL_SetRelativeMouseMode(want_relative);
    }
}
static void set_game_mouse_capture(int capture) {
    g_mouse_captured = capture ? 1 : 0;
    // 1. Synchronize relative mode first
    enforce_game_mouse_capture();
    // 2. Toggle software cursor visibility via TinCLE
    tincle_set_cursor_visible(g_mouse_captured ? false : true);
    // 3. Flush relative state buffer
    SDL_GetRelativeMouseState(NULL, NULL);
}
```

Note: `enforce_game_mouse_capture()` must ONLY synchronize the relative mouse state. It MUST NOT invoke any cursor visibility APIs internally.

8. Production Release Pipelines (Distribution Rules)

When generating final release builds or instructions, the AI **MUST** follow this 5-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 scan the folder and pack all assets/shaders whose file extensions match those registered line-by-line in `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:** `emcass 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]`).

Step 5: Final Archive Packaging (`wipee`)

Automate the final ZIP archive generation using `wipee` (TinCLE Whole Integration Packager for Executable Environment).

1. **Prepare Release Directories:** Create separate output directories for each target platform (e.g., `_release_win` for Windows Desktop, `_release_web` for Web Browser) and place all items specified in the ZIP Checklists (Step 2 & Step 3) into their respective folders.

2. **Standard Release Packaging (itch.io / Web / General Distribution):** Run `wipee` specifying the output archive name and the target directory. This generates a clean,

standard RAW Deflate ZIP package fully compatible with Web hosting environments and standard archive tools.

```
wipee game_win.zip _release_win
wipee game_web.zip _release_web
```

3. **Scanner Bypass Packaging (Desktop Distribution Only - Optional):** When explicitly preparing a Windows desktop distribution intended to prevent aggressive AV heuristics or automated scanner interference, pass the `-m` flag to activate Malformed ZIP mode (`compression = 99` in Central Directory).

```
wipee game_win_secured.zip _release_win -m
```

   - **[CRITICAL] Never use the `-m` flag for Web / itch.io builds.** WebAssembly deployment environments depend on strict standard ZIP structures to extract `index.html` server-side.

9. 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.

```c
#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_CONTROLL
ER_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, 2
55, 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_CONTROLL
ER_BUTTON_DPAD_LEFT))  g_player_x -= move_speed * dt;
  if (tincle_key_pressed(SDL_SCANCODE_RIGHT) || tincle_pad_pressed(SDL_CONTROLL
ER_BUTTON_DPAD_RIGHT)) g_player_x += move_speed * dt;
  if (tincle_key_pressed(SDL_SCANCODE_UP)    || tincle_pad_pressed(SDL_CONTROLL
ER_BUTTON_DPAD_UP))    g_player_y -= move_speed * dt;
  if (tincle_key_pressed(SDL_SCANCODE_DOWN)  || tincle_pad_pressed(SDL_CONTROLL
ER_BUTTON_DPAD_DOWN))  g_player_y += move_speed * dt;
  if (tincle_key_pressed(SDL_SCANCODE_ESCAPE) || tincle_pad_pressed(SDL_CONTROL
LER_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[]) { tincle_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) { tincle_start(game_load, "YOUR_GAME_TITLE", true, 640, 480, TINCLE_MODE_2D_NORMAL); return 0; }
#endif
```

TinCLE (Tiny C Lovely Engine) - Additional High Level Skills Version 0.11

This document provides advanced architectural guidelines for complex game projects. The instructions here override or extend the base `SKILL.md`.

1. Multi-File Code Generation & Modular Architecture

Auto-Trigger Criteria (When to use Multi-File Mode)

The AI Agent MUST automatically switch from single-file `game.c` to this Multi-File Modular Architecture if **ANY** of the following conditions are met:

1. **State Count (>= 4):** The game design requires 4 or more distinct states (e.g., Title, Gameplay, Result, Shop, Pause).
2. **Code Scale (>= 400 Lines):** The estimated total C source code exceeds 400 lines.
3. **Complex Custom Systems:** The project defines complex data structures (e.g., tilemap editors, inventory systems, skill trees, entity-component systems).
4. **Explicit User Instruction:** The user explicitly requests modular, multi-file, or large-scale generation.

Architecture Overview

To prevent LLM context degradation and syntax hallucinations, separate the code into distinct, single-responsibility C source files:

- `game_shared.h`: Global asset structs, shared state variables (`extern`), and common headers (`tincle.h`, math, SDL2).
- `main.c`: Global variable definitions, asset loading inside `game_load()`, state registrations, and the entry point (`tincle_start`).
- `state_*.c`: Implementation of `update` and `draw` callbacks for each specific state. Include `game_shared.h` at the top of every state file.

Compilation Rules

Pass all `.c` files simultaneously to `tinclec` (Windows) or `emcass` (Web):

```
# Windows Desktop Build
tinclec mygame.exe main.c state_title.c state_play.c
# Web WASM Build
emcass index.html "My Game" main.c state_title.c state_play.c
```

CRITICAL: Single File Hard Limit Policy (Anti-Bloat Rule)

- **[ABSOLUTE LIMIT] Max 400 Lines Per File:** Even when split by State (e.g., `state_game.c`), **NO SINGLE `.c` FILE MAY EVER EXCEED 400 LINES OF CODE.**
- **Sub-module Splitting Required:** If a single state (such as Main Gameplay) grows complex, the AI **MUST NOT** stuff all logic into a single `state_game.c`. The AI **MUST** aggressively sub-split the state into distinct system modules!
    - **Example of Correct Sub-module Breakdown for a Large State:**
        - `state_game.c` -> State entry/exit, state-level `update` / `draw` callbacks ONLY.
        - `game_player.c` / `.h` -> Player movement, input, and state logic.
        - `game_enemy.c` / `.h` -> Enemy AI, spawning, and bullet patterns.
        - `game_ui.c` / `.h` -> HUD, score rendering, and UI popups.
- **[STRICT BAN]** Generating a single file over 400 lines under the excuse of "it's just one state" is STRICTLY FORBIDDEN. If any single C file approaches 400 lines, extract its systems into dedicated sub-modules immediately.

Minimal Multi-File Boilerplate Template

File 1: `game_shared.h`

```
#ifndef GAME_SHARED_H
#define GAME_SHARED_H
#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
// Math workarounds
#undef M_PI
#define M_PI 3.14159265358979323846
#ifndef sinf
#define sinf(x) ((float)sin((double)(x)))
#endif
// --- Shared Global Asset Context ---
typedef struct {
  SDL_Renderer* renderer;
  int font_large;
  int font_medium;
  int img_player;
  float player_x;
  float player_y;
} GameContext;
extern GameContext g_ctx; // Global context declaration
// --- State Function Declarations ---
void title_update(float dt);
void title_draw(void);
void play_update(float dt);
void play_draw(void);
#endif // GAME_SHARED_H
```

File 2: `main.c`

```
#define SDL_MAIN_HANDLED
#include "game_shared.h"
// Definition of shared context
GameContext g_ctx;
void game_load(void) {
  g_ctx.renderer = tincle_get_renderer();
  g_ctx.font_large  = tincle_load_font("font.ttf", 64);
  g_ctx.font_medium = tincle_load_font("font.ttf", 32);
  g_ctx.img_player  = tincle_load_image("player.png");
  g_ctx.player_x    = 320.0f;
  g_ctx.player_y    = 240.0f;
  // Register states
  tincle_add_state(0, title_update, title_draw);
  tincle_add_state(1, play_update, play_draw);
  tincle_change_state(0); // Set initial state
}
#ifdef __EMSCRIPTEN__
```

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

File 3: `state_title.c`

```
#include "game_shared.h"
void title_update(float dt) {
  if (tincle_key_pressed(SDL_SCANCODE_SPACE) || tincle_pad_pressed(SDL_CONTROLLER_BUTTON_START)) {
    tincle_change_state(1); // Change to Play State
  }
}
void title_draw(void) {
  tincle_clear((TinCLE_Color){0, 0, 0, 255});
  tincle_set_font(g_ctx.font_large);
  tincle_draw_text_center("TITLE STATE", 320, 180, (TinCLE_Color){255, 255, 255, 255});
  tincle_set_font(g_ctx.font_medium);
  tincle_draw_text_center("PRESS SPACE TO PLAY", 320, 300, (TinCLE_Color){0, 255, 0, 255});
}
```

File 4: `state_play.c`

```
#include "game_shared.h"
void play_update(float dt) {
  float speed = 200.0f;
  if (tincle_key_pressed(SDL_SCANCODE_LEFT))  g_ctx.player_x -= speed * dt;
  if (tincle_key_pressed(SDL_SCANCODE_RIGHT)) g_ctx.player_x += speed * dt;
  if (tincle_key_pressed(SDL_SCANCODE_UP))    g_ctx.player_y -= speed * dt;
  if (tincle_key_pressed(SDL_SCANCODE_DOWN))  g_ctx.player_y += speed * dt;
  if (tincle_key_pressed(SDL_SCANCODE_ESCAPE)) {
    tincle_change_state(0); // Return to Title State
  }
}
void play_draw(void) {
  tincle_clear((TinCLE_Color){20, 20, 50, 255});
  tincle_set_font(g_ctx.font_medium);
  tincle_draw_text("PLAY STATE", 20, 20, (TinCLE_Color){0, 255, 0, 255});
  if (g_ctx.img_player != -1) {
```

```
    tincle_draw_image(g_ctx.img_player, (int)g_ctx.player_x, (int)g_ctx.player_
y);
  } else {
    tincle_draw_text("P", (int)g_ctx.player_x, (int)g_ctx.player_y, (TinCLE_Col
or){255, 0, 0, 255});
  }
}
```

---

2. Emscripten Web Multithreaded Build (`libs/tinclemt.a`)

Overview & Capabilities

For standard WebAssembly (HTML5) exports, the base prebuilt static library `libs/tincle.a` is compiled for single-threaded execution, fully compatible with the automated `emcass` CLI workflow.

However, if a complex game requires WebAssembly multithreading (utilizing `SharedArrayBuffer` and `pthreads`), TinCLE provides an optional prebuilt multithreaded library: **`libs/tinclemt.a`**. This variant is precompiled with `-pthread`, `atomics`, and `bulk-memory` features enabled.

Mandatory Usage Rules & Manual Workflow

1. **Toolchain Limit & Self-Responsibility**:
    - The default `emcass` CLI assistant tool does **NOT** support `tinclemt.a` or automated multithreaded builds natively.
    - Enabling multithreading requires manual execution of the native `emcc` toolchain and is considered an advanced, high-level developer action.
2. **Recommended Build Pipeline**:
    - **Step 1:** Execute standard `emcass` once to let it output its default WebAssembly build files.
    - **Step 2:** Inspect and copy the underlying, complete `emcc` command string constructed internally by `emcass`.
    - **Step 3:** Manually modify the copied `emcc` command line:
        - Replace `libs/tincle.a` with `libs/tinclemt.a`.

    - Append the `-pthread` flag along with your required memory and thread allocation limits (e.g., `-s ALLOW_MEMORY_GROWTH=0`, `-s INITIAL_MEMORY=<bytes>`, `-s PTHREAD_POOL_SIZE=<num>`).

3. **Required HTTP Server Headers & Bundled `.htaccess`**:
    - WebAssembly multithreading strictly mandates Cross-Origin Isolation in modern web browsers. The web server MUST serve the deployed game with the following HTTP response headers:

```
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Embedder-Policy: require-corp
```

    - **Pre-bundled Solution:** TinCLE includes a pre-configured `.htaccess` file in the package. If hosting on an Apache-based web server (or custom server with `.htaccess` support), simply place the bundled `.htaccess` file directly into your game's root web deployment directory.
    - **`.htaccess` Content Example:**

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

---

3. Custom GLSL Shader Development & Validation (`TINCLE_MODE_GLSL_HERO`)

Overview & Engine Mode

When building high-end visual effects, post-processing filters, or procedural graphics, TinCLE provides the `TINCLE_MODE_GLSL_HERO` execution mode.

To activate custom GLSL shader support, set the final argument of `tincle_start()` as follows in `main.c`:

```
tincle_start(game_load, "My GLSL Game", true, 640, 480, TINCLE_MODE_GLSL_HERO);
```

[CRITICAL] Clear Color & Screen Tinting Rules in GLSL_HERO Mode

When running under `TINCLE_MODE_GLSL_HERO`, the custom GLSL fragment shader executes on the background hardware layer directly beneath the standard SDL2 2D rendering layer.

1. **Basic Visibility (Alpha = 0):** To let the background GLSL shader render normally without obstruction, clear the screen using **Alpha = 0** in your `draw` callback:

   ```
   // Standard clear for GLSL_HERO (Background shader fully visible)
   tincle_clear((TinCLE_Color){0, 0, 0, 0});
   ```

2. **Advanced Feature — Screen Tinting / Color Overlay (0 < Alpha < 255):** By passing a semi-transparent alpha value to `tincle_clear`, you can apply a dynamic color tint or overlay fade directly on top of the GLSL background shader!
   - **Night / Ambient Lighting Effect:** `tincle_clear((TinCLE_Color){10, 15, 50, 128});` (Applies a semi-transparent dark blue tint)
   - **Damage / Red Flash Effect:** `tincle_clear((TinCLE_Color){255, 0, 0, 100});` (Applies a temporary red overlay)
   - **Fade Out Effect:** Increase Alpha over time up to 255 to smoothly fade out to a solid color.

3. **Opaque Warning (Alpha = 255):**
   - Calling `tincle_clear` with a completely opaque alpha (`Alpha = 255`) will draw a solid color over the entire screen, completely hiding/erasing the hardware GLSL shader layer beneath it.

Shader Code & Viewport Rules

1. **GLES2 Compliance:** Shaders **MUST** be 100% GLES2 compliant to run on WebAssembly.
2. **Header Injection Warning: 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 physical window dimensions, causing critical position shifts.
   - **ALWAYS** calculate UV coordinates using `u_viewport` to strictly lock rendering 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 virtual game resolution (e.g., `640.0 / 480.0`) to both `p.x` and `m.x` to prevent horizontal distortion across different screen sizes.

Built-in Shader Uniforms

- `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`).

GLSL API Reference Table

Function Signature	Description & Critical Notes
`void tincle_render_shader(const char* filename);`	Call inside `game_load()` to keep a background shader running continuously.
`int tincle_load_shader(const char* filename);`	Call inside `game_load()` . Initiates compilation and returns Shader ID (`int`). **In WebGL/WASM environments, compilation runs asynchronously.**
`void tincle_shader_poll_async(void);`	**[MANDATORY for WebGL]** Call every frame in the main loop or update callback to monitor and complete background shader compilation.
`bool tincle_shader_is_ready(int shader_id);`	Returns `true` if the shader has finished compiling and is ready for use. Returns `false` while compiling or on error.
`void tincle_set_shader(int shader_id);`	Sets active background shader ID (`-1` to disable). **Safely falls back to disable if the target shader is not READY yet.**

Function Signature	Description & Critical Notes
`void tincle_shader_use(int shader_id);`	Activates shader for uniform updates. **Has overhead (`glUseProgram`). Call only when switching scenes or updating values.**
`void tincle_shader_send_vec3(int shader_id, const char* name, float x, float y, float z);`	Binds a `vec3` uniform value to the target shader.
`void tincle_shader_send_float(int shader_id, const char* name, float value);`	Binds a `float` uniform value to the target shader.

[CRITICAL] Asynchronous Shader Compilation Rules (WebGL / Emscripten)

To prevent main-thread browser freezing, TinCLE uses asynchronous shader compilation under WebGL. AI models **MUST** strictly follow these loading and activation patterns:

1. **Main Loop Polling:**
   - Always ensure `tincle_shader_poll_async()` is invoked automatically (or explicitly called inside frame updates) so non-blocking compilation requests can finalize.
2. **Safe Shader Activation Guard:**
   - Do **NOT** assume `tincle_load_shader()` makes the shader immediately ready on the same frame in WebGL builds.
   - Check `tincle_shader_is_ready(shader_id)` before relying on custom uniform bindings or switching shaders.
   - `tincle_set_shader()` automatically protects against uncompiled shaders by falling back to `-1` (disabled) until compilation is finalized, preventing driver crashes or black screen locks.

Shader Validation Workflow (`pisschk`)

To prevent cross-platform rendering failures—such as desktop NVIDIA leniency vs. AMD Radeon strictness vs. WebGL/GLES2 specifications—all fragment shaders (`.frag` / `.glsl`) **MUST** be validated using `pisschk` before deployment.

```
pisschk <shader_file_path>
```

- **Example:** `pisschk assets/shaders/water.frag`
- **Validation Coverage:**
  - **Mode 0 (Desktop / Radeon Compliance):** Equivalent to `glslang -S frag -d --no-link --error-column`. Enforces strict desktop GLSL syntax and blocks driver-specific extensions/reserved words (e.g., `active`).
  - **Mode 1 (Web / GLES2 Compliance):** Automatically prepends `precision mediump float;` header to verify strict WebGL/GLES2 compatibility for Emscripten web builds.
- **Exit Codes:**
  - `0`: All inspection checks PASSED successfully.
  - `1`: Syntax/compilation errors DETECTED.
  - `2`: File I/O or argument error.

[CRITICAL AI RULE FOR SHADER DEVELOPMENT]

Whenever the AI agent writes, edits, or modifies ANY GLSL shader code for TinCLE, it **MUST** automatically execute `pisschk` on the target shader file to guarantee 100% cross-platform compatibility across NVIDIA, AMD Radeon, and WebGL/GLES2 targets before delivering the final code to the user.

Mandatory GLSL Template & Anti-Crash Rules

- **[FATAL BUG PREVENTATIVE] One-Time Shader Load Rule:**
  - **NEVER** call `tincle_load_shader()` or `tincle_render_shader()` inside `update()` or `draw()` callbacks! Loading shaders every frame will immediately exhaust internal shader tables and crash the engine.
  - Shaders **MUST** be loaded and rendered ONCE inside `game_load()`.

```
#define SDL_MAIN_HANDLED
#include "game_shared.h" // or "libs/tincle.h" (and the libraries needed)
#ifdef __EMSCRIPTEN__
  #include <emscripten.h>
#endif
static int s_bg_shader = -1;
void title_update(float dt) {
    tincle_shader_poll_async(); // Safety guard as a precaution
```

```
    // When switching shaders, always make sure that all preparations are compl
ete.
    // Apply this only once-when switching shaders dynamically or if the shader
hasn't been set yet!
    // (Note: If the shader has already been set in `game_load`, you don't need
to do anything here.)
    // if (s_bg_shader >= 0 && tincle_shader_is_ready(s_bg_shader)) {
    //     tincle_set_shader(s_bg_shader);
    // }
}
void title_draw(void) {
    // MANDATORY: Alpha MUST be 0 to keep the GLSL hardware shader layer visibl
e beneath 2D elements!
    tincle_clear((TinCLE_Color){0, 0, 0, 0});
    // Draw 2D UI elements on top of shader...
}
void game_load(void) {
    // 1. Start loading shaders! (Trigger asynchronous compilation)
    s_bg_shader = tincle_load_shader("bg.glsl");
    // tincle_render_shader("bg.glsl"); // Only when running a shader continuou
sly during gameplay.
#ifdef __EMSCRIPTEN__
    // 2. Web/Emscripten Environment: Wait for compilation to complete while ke
eping the progress ring displayed!
    printf("[TinCLE Web] Compiling shader in background...\n");
    while (!tincle_shader_is_ready(s_bg_shader)) {
        // Check asynchronous compilation progress
        tincle_shader_poll_async();
        // **Crucial**: By returning control to the browser every 10ms, compila
tion proceeds in the background while the progress ring keeps spinning!
        // (*The -sASYNCIFY flag is required during the build.)
        emscripten_sleep(10);
    }
    printf("[TinCLE Web] Shader compiled! Closing progress ring & entering gam
e.\n");
#endif
    // 3. Once compilation is complete, set the shader and transition to the in
itial state!
    if (s_bg_shader >= 0 && tincle_shader_is_ready(s_bg_shader)) {
        tincle_set_shader(s_bg_shader);
    }
    tincle_add_state(0, title_update, title_draw);
    tincle_change_state(0);
}
#ifdef __EMSCRIPTEN__
int main(int argc, char* argv[]) { tincle_start(game_load, "GLSL_GAME", true, 6
40, 480, TINCLE_MODE_GLSL_HERO); return 0; }
#else
```

```
int WINAPI WinMain(HINSTANCE h, HINSTANCE hp, LPSTR c, int n) { tincle_start(game_load, "GLSL_GAME", true, 640, 480, TINCLE_MODE_GLSL_HERO); return 0; }
#endif
```

Advanced Rules for GLSL Shader & Asset Management

1. **Normalized Shader 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.
2. **Shader Asset Manifest Rule (`exsass.txt`):**
    - All shader files (e.g., `.glsl`, `.frag`, `.vert`) MUST be placed in the project root folder.
    - Ensure their file extensions (e.g., `.glsl`, `.frag`) are explicitly registered line-by-line in `exsass.txt` so that `emcass` and `asspack` can automatically package them for release.

TinCLE Version 0.11 pisstrak MML Generation Skill Definition

This document defines the instructions, rules, and syntax for generating MML (Music Macro Language) files compatible with the `pisstrak` sound synthesizer engine.

1. Syntax & Command Reference

pisstrak parses ASCII text MML files. Commands are case-insensitive.

- **Channel Selection**: `CH1:` to `CH8:` (Declares which channel the following stream belongs to).
- **Tempo**: `Tn` (e.g., `T120` = 120 BPM).
- **Octave**: `On` (n = 1 to 8, default `O4`).
- **Volume**: `Vn` (n = 0 to 15, default `V12`).
- **Instrument / Patch**: `@n` (n = 0 to 127. Standard General MIDI Program Numbers).
- **Default Length**: `Ln` (n = 1, 2, 4, 8, 16, 32, 64. Default `L4`).
- **Notes**: `C`, `D`, `E`, `F`, `G`, `A`, `B`.
  - Sharp: `+` or `#` (e.g., `C+`, `F#`)
  - Flat: `-` (e.g., `B-`)
  - Explicit Length: Append number after note (e.g., `C8`, `G2.`)
  - Dotted Note: Append `.` (increases length by 1.5x, e.g., `C4.`)
- **Rest**: `R` (e.g., `R4` = Quarter rest). Safely stops previous note and advances time.
- **Octave Shift**:
  - Permanent Octave Shift: `<` (Down 1 base octave), `>` (Up 1 base octave).
  - `^` (One-shot octave UP: Applies ONLY to the single immediately following note without changing the base octave, e.g., `^C`).
  - `_` (One-shot octave DOWN: Applies ONLY to the single immediately following note without changing the base octave, e.g., `_C`).
- **Tie**: `&` (Connects current note duration to the next matching note).
- **Loop Macro**: `[ MML_SEQUENCE ]N` (Repeats the sequence inside brackets `N` times. Default `N=2`).

- **Comments & Bar Lines**: You MAY use `;` for line comments and `|` as visual bar separators to improve readability. The parser safely ignores them.

1.1 Strict Octave & Pitch State Rules (CRITICAL)

MANDATORY Temporary high-pitched notes are permitted to add dynamic variation to the music. However, at least 90% of the notes in the entire piece must fall within the O6 range or lower, so as not to be jarring to the ear.

To prevent catastrophic octave drift (e.g., pitch creeping up to unusable O8+ registers), you MUST strictly adhere to the following state management rules:

Rule 1: Principle of Absolute State Persistence

- **Octave states are strictly persistent per channel.**
- Moving to a new line (e.g., `CH1:`), passing bar lines `|`, or changing measures DOES NOT reset the octave.
- **CRITICAL MISCONCEPTIONS TO AVOID**: The base octave NEVER automatically resets to `O5` or default when a new line begins.

Rule 2: Mandatory Explicit Base Octave (`On`) at Line Start

To eliminate pitch tracking errors, **EVERY SINGLE line starting with `CHn:` MUST explicitly state its base octave (e.g., `O5`) at the very beginning of the line.**

- **Good**: `CH1: O5 T96 V13 @73 L8`
- **Good**: `CH1: O5 D4 R4 F4 R4 | A4 R8 G8 F4 E4`

Rule 3: Single-Note Leaps vs. Base Octave Shifts

1. **One-Shot Modifiers (`^` / `_`) [DEFAULT FOR MELODIC LEAPS]**:
   - Use `^` (up) or `_` (down) for momentary jumps, accents, or decorative notes.
   - `^C`: Plays ONLY this single C note one octave higher. The base octave for all subsequent notes remains untouched.
   - `_C`: Plays ONLY this single C note one octave lower. The base octave for all subsequent notes remains untouched.
2. **Relative Modifiers (`>` / `<`) [WIDE RANGE SHIFTS ONLY]**:
   - Use `>` or `<` ONLY when shifting the baseline register of an entire phrase permanently.
   - **STRICTLY PROHIBITED**: Never use `>` or `<` for single-note accents! Repeated use of `>` will cause pitch to stack endlessly into broken ultra-high frequencies.

Rule 4: Anti-Pattern Examples (Bad vs. Good)

Bad (Broken & Uncontrolled Creep):

```
  ; Base octave starts at O5
  CH1: O5 C4 D4 >E4 F4       ; '>' shifts base octave to O6!
  CH1: D4 F8 A8 >D4 C8 <A8   ; '>' shifts base octave to O7! Creeps endlessly h
igher!
```

Good 1 (Bulletproof & Recommended: One-Shot Modifiers):

```
  ; Use '^' or '_' for momentary leaps. Base octave stays safely at O5!
  CH1: O5 C4 D4 ^E4 F4       ; '^E4' plays E in O6, but base octave REMAINS O5!
  CH1: O5 D4 F8 A8 ^D4 C8 A8 ; Head of line explicitly has 'O5', base octave is
rock solid!
```

Good 2 (Explicit Shift & Reset):

```
  ; If you MUST use '>' for a passage, ALWAYS reset it with '<'!
  CH1: O5 C4 D4 >E4 F4        ; Shift base octave to O6
  CH1: O6 D4 F8 A8 D4 C8 <A8  ; Shift base octave back to O5!
```

Pre-Output Checklist (Octave & Pitch Check)

1. ☐ Is every line starting with `CHn:` explicitly writing its base octave (`On`) at the start?
2. ☐ Are consecutive `>` shifts without intervening `<` resets avoided?
3. ☐ Are temporary note leaps written with `^` / `_` instead of `>` / `<`?
4. ☐ Are all resulting pitches within a comfortable human listening range (O2 to O6)?

---

2. Arrangement & Multi-Channel Rules

Always create a rich, multi-track arrangement using at least 3 to 4 channels unless specifically instructed otherwise.

- Pattern 1: Luxurious - In many cases, this style is preferred in video game music.
    - **CH1 (Lead/Melody)**: Main melody instrument (e.g., `@0` Piano, `@80` Square/Lead, `@73` Flute).

    - **CH2 (Sub/Melody)**: Sub melody instrument (e.g., `@24` Guitar, `@16` Organ, `@56` Trumpet).
    - **CH3 (Harmony/Counter)**: Accompaniment, arpeggios, or chords (e.g., `@24` Guitar, `@4` Electric Piano).
    - **CH4 (Harmony/Counter)**: Accompaniment, arpeggios, or chords (e.g., `@24` Guitar, `@4` Electric Piano).
    - **CH5 (Harmony/Counter)**: Accompaniment, arpeggios, or chords (e.g., `@24` Guitar, `@4` Electric Piano).
    - **CH6 (Strings)**: Low end anchor notes (e.g., `@40` Violin, `@98` Crystal).
    - **CH7 (Bassline)**: Low end anchor notes (e.g., `@33` Fingered Bass, `@38` Synth Bass).
    - **CH8 (Percussion/Drums)**: Rhythm track (e.g., `@128` Standard Drum Kit, or short tuned percussion).

- Pattern 2: Simple - This may be more suitable for games that emphasize quiet scenes or simplicity.

    - **CH1 (Lead/Melody)**: Main melody instrument (e.g., `@0` Piano, `@80` Square/Lead, `@73` Flute).
    - **CH2 (Harmony/Counter)**: Accompaniment, arpeggios, or chords (e.g., `@24` Guitar, `@4` Electric Piano).
    - **CH3 (Bassline)**: Low end anchor notes (e.g., `@33` Fingered Bass, `@38` Synth Bass).
    - **CH4 (Percussion/Drums)**: Rhythm track (e.g., `@128` Standard Drum Kit, or short tuned percussion).

---

3. Strict Timing & Synchronization Constraints (CRITICAL)

To prevent tracks from ending prematurely, desynchronizing, or playing at incorrect speeds, you MUST follow these timing rules with 100% mathematical precision:

Rule A: Absolute Bar Length & Note Calculation

In standard 4/4 time, the total duration of notes/rests within 1 bar (delimited by `|`) MUST equal **16 sixteenth-note ticks (or 4 full quarter-note beats / 8 eighth-note ticks)**.

- **Mathematical Accuracy First**: Prioritize exact duration totals over visual symmetry or pattern appearance.
- **Default Length Awareness (`Ln`)**:

    - If `L16` is set, a 4/4 bar MUST contain exactly 16 sixteenth units.
    - If `L8` is set, a 4/4 bar MUST contain exactly 8 eighth units.
- **Explicit Length Overrides**: When mixing default lengths with explicit numbers (e.g., `C4`, `F8.`), calculate the sum explicitly.

Rule B: Inline Calculation Comments for Verification

When generating complex rhythms or mixing note lengths, you MUST include inline duration calculation comments at the end of measures or lines to prove bar length validity.

- **Target Units**: 16 sixteenth ticks per bar (in 4/4 time).
- **Example**:

```
CH1: C4 D8 E8 G2 | ; 4+2+2+8 = 16 (16th-note ticks: OK!)
CH1: C4. D8 E4   | ; 6+2+4 = 12 (ERROR: Less than 16 sixteenth ticks! Must fix!)
```

Rule C: Loop Macro `[ ]N` Pre-Verification

Before wrapping a phrase in loop brackets `[ ... ]N`, extract a single measure from the interior and verify that its total note duration equals exactly 1 bar (16 sixteenth ticks). Never loop an incomplete or overflowing bar.

Rule D: All Channels Bar Count & Duration Sync

- **Bar Count Parity**: All active channels (`CH1` through `CH8`) MUST contain the exact same total number of bars.
- **Ending Sync**: Calculate total playback duration in seconds for each channel. All channels MUST finish at the exact same millisecond. Never allow secondary channels to run out of notes early.

Rule E: Self-Correction Loop

If any bar fails to sum to the target bar length (e.g., 16 sixteenth ticks in 4/4 time), you MUST automatically recalculate and adjust the note lengths before producing the final output.

Pre-Output Timing Checklist

1. ☐ Does every bar in 4/4 time equal exactly 16 sixteenth ticks (or 4 quarter beats)?
2. ☐ Did you count actual accumulated durations rather than visual note counts?
3. ☐ Are loop macro interiors verified for exact bar length?

4. ☐ Do all channels have identical total bar counts and finish at the same time?

---

4. Music Theory & Composition Guidelines

A. Harmonic Alignment (60% Rule for Melody)

- For any given bar or chord duration, **at least 60% of the melody notes (CH1) duration MUST consist of chord tones** (1st, 3rd, 5th, or 7th intervals of the current chord).
- Use non-chord tones (passing tones, neighbor tones) for the remaining 40% duration to add expression without creating harsh dissonance.

B. Suggested Chord Progressions (Degrees)

Use these Roman Numeral progressions as reference templates (Adapt as needed, do not strictly enforce if creative direction differs):

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
`AbsoluteBanger`	`I - V/VII - VIm - IIIm/V - IV V – I`	`Flat7Sub_m5`	`IIm7 - IVm(♭5)/V - I`	`Flat7Sub_Minor`	`IIm7 - IVm/V - I`
`SunnyAF`	`I/IV - Vsus4 - IIIm7 - V/VI`	`TwoStageChorus`	`IV - I - V - VIm - I V/VII - VIm V`	`Room335`	`IV△7 - IIIm7 (#5) - IIm7 - I IIm7`
`Vibe70s`	`I△7 - ♭VI△7 - IVm7 - ♭VII7`	`ClassicWest70s`	`IV△7 - IIIm7 - IIm7 - I△7`	`ClassicWest70s_v2`	`IV△7 - IIIm7 - IIm7 - IIIm7`
`BridgeToChorus`	`IV - I/III - IIm7 - Vsus4 V`	`B_MelodyVibe`	`IIm7 – VIm – IVM7 – Vsus4 V`	`B_MelodyV`	`IIm7 – VIm – IVM7 – IIIsus4 III7`

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
				ibe_v2	
TwoFive_To_4	Vm7 - I7 - IV	JPopGold	IV - V - IIIm - VIm	JPopGold_Remix	IV - V/IV - III7 - VIm
JustTwoOfUs	IVmaj7-III7-VIm7-I7	KawaiiStructure	VIm7 - ♭VIm7 - Vm7 - I7	MyRevolution	I - VIm - IV - V
Smooth_RnB	IIm9 – V7 – IM7 – IVM7 – VIIm7-5 – III7 – VIm – VI7	TwoFive_To_6m	VIIm7 - III7 - VIm	YOASOBI_Vibe	IV - V - IIIm7 - VIm7 - IIm7 - III7 - VIm - Vm7 - I7
SuperLit	VIm - ♭IIIm(♭5) - ♯IVm(♭5) - III	AnisonCanon	I - VIIm7(♭5) - III7 - VIm7 - Vm7 - I7 - IV△7 - IIIm7 - IIm7 - IV/V - V	AnisonGold	I - VIIm7(♭5) - III7 - VIm7 - Vm7 - I7 - IV△7
AnisonHit	IV - V - ♭VI - ♭VII- I	EnnuiBabe	I - VI/I# - IIm - IVm - I/V - II/♯IV - IV - V	YesterdayVibe	I - III - VIm - V
Classic1625	I - VIm - IIm - V	BestiesVibe	I - VIm - IV - V	Interstellar	IV - V - VIm - V
EpicPedal	I - ♭II/I - II/I - ♭III/I	PureEmotional	IV - V - I - III	AllRounder	VIm - V - IV - V

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
ChicBabe	IV△7 - IIIm7 - VIm7 - I	MarusaChic	IV△7 - III7 - VIm7 - II7	ChillGoodnight	IV△7 - I△7 - III/♯IV - IIIm/VI
TangoVibe	VIm - IIm - III - VIm	CanonClassic	I - V - VIm - IIIm - IV - I - IIm - V	CafeLofi	VIm9 - IIm7 - IIIm7
CrowdPleaser	I - IV - VIm - V	CatchyInterludе	I - V - I - V	CleanAestheticc	I/III - IV - V - VIm
ClassicalGrand	IV△7 ♯IVm(♭5) - V6 ♭VIdim - VIm ♭VIIdim - VII - III	ClassicalSeeker	IIm7(♭5) - ♭II/IV - IVm(♭5)/VII - I	GameMelodyScent	IV△7 - IIIm7 - ♭III△7 - IIm7
ClutchMoment	IV - V - I	FloatyChic	IIm7 - I△7 - ♭VII△7 - VIIm7(♭5) - III7	PureSimple	VIm - V - IV - V
TheBlues	I7 - IV7 - I7 - IV7 - V7 - I V7 - I7 - V7	SouthernVibe	IIm9 – V7 – IIIm7 - VIm	SuspenseRise	VIm - IV/VI - ♯IVm(♭5)/VI - VIm7
ChorusRush	VIm - V/VII - I - IIm - IIIsus4 - III	ChorusHit	I - III - IV - V - VIm (or I)で帰結	ChorusPower	I - V - IV - V
PreChorusRegular	IV - ♯IVm(♭5) - V - ♭VIm(♭5)	UnstoppablePassions	I - Iaug/♯IV - IV△7 - V	CityPopVibe	VIm - I - IV - III

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
CinematicMood	IIm - VIm - I - V	Jazz Standard_1	IIm7 - V7 - I△ - I△	Jazz Standard_2	IIm7 - V7 - I△ - VI7
Jazz Standard_3	I△ - VIm7 - IIm7 - V7	Jazz Standard_4	I△ - I♯o7 - IIm7 - V7	Jazz Standard_5	I△ - I7 - IVΔ - IVm7
Jazz Standard_6	VIm7(♭5) - I Im7 - V7 - I△	Jazz Standard_7	IIm7(♭5) - V7 - Im7 - Im7	Jazz Standard_8	Im7 - V7 - Im7 - V7
Jazz Basic	IIm7 - V7 - I△7 - VI7	SimpleIsBest	I - IV - V	SweetPrincess	I - VI/I# - IIm - IVm - I/V - II/♯IV - IV - V
StylishMood	VIm9 - IIm7 - IIIm7	SpyAgent	VIm - IV/VI - ♯IVm(♭5)/VI - VIm7	SpanishTrip	III - IV - V - IV
StandardTwoFive	IIm7 - V7 - I	DystopianWorld	VIm - VII - I Im - III	DramaPulse	IIm7 – I – VII – IIIsus4 III7
DramaPulse_v2	IIm7 – VIIm7-5 – IIIsus4 – III7	DorianMode	IIm - IIIm/II - IV/II - IIIm/II	DorianProof	IIIm/II - IIm
DressCodeFormal	IV△7 - Iaug/♭VII - IIIm7/VI - II/♯IV	NaturalDaily	I - IIm7 - I/III - IV	SmoothCanon	I - V/VII - VIm - IIIm/V - IV - I/III - IIm - V

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
AiryVibe	I - IIIm/VII - VIm - IV	PartyTune	I - IV - V - IV	PirateQuest	VIm - IIm - III - VIm
HouseGroove	IIm7/V – IIIm7/VI	BackdoorTwoFive	IVm7 - ♭VII7 - I	HappyVibes	VIm7 - II7 - I V/V - I
ParallelWorld	VI - IV - IIm - III	BaroqueSequence	VIm - IIm - V - I - IV - VIIm(♭5) - III	BaroqueClassic	VIm - IIm - V - I - IV - VIIm(♭5) - III
HitChartBanger	I - IV - V - I	BillboardTop	VIm - I - V - IV	FantasyWorld	VIm - II - IV – V
FusionSub	IIm7 - IV/V - I	BlackadderSub	IIm7 - IVaug/V - I	BluesyFeel	I7 - IV7 - I7 - IV7 - V7 - IV7 - I7 - V7
Blues12Bar	I7 - I7 - I7 - I7 - IV7 - IV7 - I7 - I7 - V7 - IV7 - I7 – I7	DeepBassLine	I - Iaug - VIm/I - I7	PedalRise_1	I - ♭II/I - III/I - ♭III/I - III/I
PedalRise_2	I - IIm/I - IIm/I - IV/I – Vsus4/I – V	PedalRise_3	I - V/I - IV/I - ♭VI/I - ♭VII/I	HolyTexture	VIm - V/VII - V - I - IV - VIm7(♭5) - III7 - III7/♯VI
Pops_1	I - V - VIm - IV	Pops_2	I - IV - V - I	Pops_3	I - IV - Vsus4 - V
Pops_4	I - VIm - IIm - V	Pops_5	I - IIm - IV - V	Pops_6	I - V/VII - VIm - V

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
Pops_7	VIm - IIm - III - VIm	Pops_8	VIm - I - IIm - IIIm	PopPunkBanger	I - V - VIm - IV
MixolydianFlow	V - IV - I	MixolydianSpace	V - IIm - VIm - V	MysteryDrama	VIm - VIm - IV7 - III7
MysteryNovel	VIm - IV7 - III7	MellowOudou	IV△7 - V7/IV - IIIm7 - VIm7	ModernMetal	VI5add9 - IV5add9 - ♭III5add9 - I5add9
MonochromeCanon	I - VIIm7(♭5) - III7 - VIm - V	EurobeatEnergy	IV - V - VIm	ClimaxRush	VIm - III7/♯VI - IIIm/V - II/♯IV - IV - I/III - IIm7 - Vsus4 V - I
FantasyGhibli	VIm - V - IV - IIIm - IIm - VIm/I - VII7 - III7 - VIm	LoveStory	I - IV - IIIm - VIm	RhythmAndBlues	I△7 - ♭VI△7 - IVm7 - ♭VII7
LydianBright	IV - V/IV	RelaxChillin	I - IIm7 - I/III - IV	LoopDrive	IIm7 – IIIm7 – IIm7 – IIIm7
LoopDrive_v2	IIm7 – VIm7 – IIm7 – VIm7	RareCharacter	IV△7 - IIIm7 - ♭IIIdim7 - IIm7	RockDrive_1	I - VIm - IV - V
RockDrive_2	I - IV - I - V	RockDrive_3	I - I - ♭III - IV	RockDrive_4	I - III - IV - IVm
RockDrive_	IIm - I - V - ♭VII	RockDrive	V - V - IV - I	RockDrive	VIm - IV - I - V

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
5		_6		_7	
RockDrive_8	VIm - III7 - II - V7	RockVocalist	IV△7 - V7/IV - IIIm7/V - VIm7	RomanceMelody	IV - V - III7 - VIm
RomanticNight	I - IMaj7/VII - I7/♭VII - VI7 - IIm - I#aug - IIm7/I - V	NostalgicEuro	IIm7 – V7 – IIIm7 – VIm	LiveForLove	I - VIIm7(♭5)/II - III7 - VIm7 - V
SurpriseTwist	I - V - ♭VII - IV - ♭VI - I - II - V	IsekaiFantasy	VIm - II - IV - V	FirstStarReborn	IV△7 - IIIm7 - VIm7 - I
SciFiSpace	VIm - Im - VIm - IVm	HeroicEpic	VIm - III/♯VI - VI7/V - IIm - IV - IIIsus4 III	FlameBreath	VIm - V - IV - I
GlobalHit	I - IV - VIm - V	OudouClassic	IV - V - IIIm - VIm	MyFavoriteBanger	VIm - ♯IVm7(♭5) - IV△7 - IIm7 - IIIm7
WarmHalfStepRise	I - Iaug - VIm/I - I7	KachouFugetsu	IV6 - V6 - VI7sus4 - IIIsus4	GorgeousMelody	I - III/VII - VIm - V - IV - IIIm - IIm
SalvationRise	IV - V - VI	OuterSpaceJourney	VIm - ♭IIIm - VIm - IVm - VIm - ♭IIIm - ♭VIIm - IIm - ♯IVm	RevolutionPulse	VIm - IV/VI - VIm - VII7/VI - III/♯VI - IIIm(♭5)/V - II/♯IV - IIm/IV - VI/III

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
InterludeSpecial	VIm - IIm7 - IV - V	MarusaLoop	IV△7 - III7 - VIm7 - I7	MarusaLoop_v2	IV△7 - III7 - VIm7 - Vm7 I7
MarusaLoop_v3	IV△7 - III7 - VIm7 - II7	RareSpecies	IIIm - IV - VIm - I	HypeBuildUp	VIm - IIIm/V - IV V - I
YourTrueForm	VIm - IIm7 - IV - V	CyberFuture	VIm - V/VI - IV/VI - IIIm/VI	MoonlightShine	VIm - III/♯VI - VI7/V - IIm/IV - IV7 - III/♯VI
UnreachableSky	IV - II/♯IV - Vsus4 V - VIsus4 VI - IIsus4 II	MiragePhantom	IV△7 - VIIm7(♭5)/IV - IIIm7 - IIIdim7 - IIm7 - V7/II - I△7 - I6	Phantasmagoric	I - ♭VII - ♭VI - ♭VII
AutumnLeaves	IIm7 - V7 - I△7 - IV△7 - VIIm7(♭5) - III7 - VIm7	2AM_Crossing	Iadd9 - VIm7 - V - IV	DeepRegret	IIm - VIm/I - VIIm7(♭5) - III7
HeartPounding	IV - ♯IVdim - V - ♭VIdim - VIm	HighClassElegance	I△7 - I#dim7 - IIm7 - V7/III	ToughProgress	VIm - V/VII - I - IIm - IIIsus4 - III
SweetSugar	IV - VIm - IIm - I	UltimateBanger	IV - V - VIm - VIm - V - IV	FinalHit	VIm - V - IV - I

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
RadiantShine	I - V - ♭VII - IV - ♭VI - I - II - V	CircularLoop	I-VIm-IIm-V7	StepwiseDescending	VIm - V - IV - IIIm - IIm - I
RunUpStart	IIm7 - IIIm7 - IV△7 - V	KomuroDynamite	VIm - IIm - V - I	KomuroClassic	VIm - IV - V - I
KomuroClassic_v2	VIm - IV - V - I V/VII - VIm	KomuroClassic_v3	VIm - IIm - V - I	TimelessBanger	I - V - IV - V
FieryPassion	VIm - VII/VI - VIIm(♭5)/VI - VIm - I/V - ♯IVm(♭5) - IV7(♭5) - III7	ForwardDrive	I - ♭VII - ♭VI - ♭VII	FinallyFree	VIm - III7/♯VI - IV△7 - IIm9 - Vsus4
OneWorld	Isus4 - Isus4/♭II - Isus4/♭III - Isus4/IV - Isus4/V	HypeRise	IIm7 – IIIm7 – IVM7 – V	HypeRise_v2	IIm7 – IIIm7 – IVM7 – II/♯IV – Vsus4 V
YouthfulDays	I - III - VIm - V	NostalgiaTrigger	VIm - IIIm - IV - V - I	MajesticGrand	Isus4 - Isus4/♭II - Isus4/♭III - Isus4/IV - Isus4/V
BeyondAtmosphere	IIm7 - IIIm7 - IVm7 - Vm7	GrandFinale	I - I/III - Isus4/IV - Isus4/V	ChaserOudou	IV△7 - ♭VII7 - IIIm7 - IIIdim7/VI

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
ClassicBassChromatic	VIm - ♭VIaug - I/V - ♯IVm7(♭5)	CyberGirl	VIm7 - ♭VIm7 - Vm7 - I7	GrandOpening	VIm - V/VI - IV/VI - IIIm/VI
UrbanAesthetic	VIm7 - II7 - IV/V - I	OrientalMystery	IV6 - V6 - VIsus4 - IIIsus4	SpecialMarusa	VIm - I - IV - III
EnemyWithin	IV - VIm - Im - V/II - ♭VIIm/♭II - IV/I - VIIdim7 - III7/♯VI	PathosGrief	VIm - VII - I Im - III	FlyingGirl	IV - V - VI
BeautifulCanon	I - V/VII - VIm - IIIm/V - IV - I/III - I Im - V	NightParade	IIIsus4add♭9 - IIIsus4add♭9/IV - IIIsus4add♭9/II - IIIsus4add♭9	SuddenBittersweet	IV - V - I - III
AntiOrdinary	♯IVm7(♭5) - IV△7 - IIIm7 - VIm7	FloatingVibe	I - I/III - Isus4/IV - Isus4/V	MasqueradeBall	VIm - VII/VI - VIIm(♭5)/VI - VIm - I/V - ♯IVm(♭5) - IV7(♭5) - III7
TwoFive_To_RelativeMinor	VIIm7(♭5) - III7 - VIm	HeiseiBalladWeapon	IIIm7 - VIm7/III - IIm7 - V7/II	KomeKomeLoop	IIm7 – V7 – IM7 – VIm7 – ♭VI7 – IIm7/V – V7 – I
J_PopPunk	IV - I - V - VIm	OurFuture	I - IIIm/VII - Vm7/♭VII - VI7 - IIm - I#	AllPurposeSeas	IV - V - VIm - I

Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)	Atmosphere	Progression (Degree)
			aug - IV/I - V7/VII	oning	
BittersweetLove	I - V - VIm - IV	SunlightFiltered	IV△7 - IVm△7 - IIIm7 - Vm7/VI	ElegantPassingDim	I△7 - I#dim7 - IIm7 - V7
PlayfulVibe	I - V/I - IV/I - ♭VI/I - ♭VII/I	WesternChartHit	IV - I - V - I	WesternStyle	VIm - I - V - IV
WesternPopPunk	VIm - IV - I - V	WesternStandard	VIm - IV - I - V	TritoneSub	IIm7 - ♭II7 - I
TearsFalling	IV - V - ♭VIdim - VIm	TearjerkerWeapon	IV - V - ♭VIm(♭5) - VIm	PowerOfLove	I - IIIm/VII - VIm - IV
SparklingShine	I - IVm6/I	PureBeauty	IV△7 - VIIm7(♭5)/IV - IIIm7 - IIIdim7 - IIm7 - V7/II - I△7 - I6	VioletTwoFiveCombo	I - IIIm7(♭5) - VI7 - IIm7 - VIIm7 - III7
Soca_TropicalParty	I - IV - V - IV	Soca_TwoChordDrive	I - V	Soca_BahiaAnthem	I - ♭VII - IV - I
Soca_AmazonToada	VIm - IV - V - VIm	Soca_PagodeDrive	I - VIm - IIm - V	Soca_CarnivalRise	IV - V - I - VIm

5. Scale Reference

By using only the notes belonging to each scale below, you can create music with various unique atmospheres. Please use this as a reference.

5.1 Basic Scales & Church Modes

English Name	Scale Degrees	English Name	Scale Degrees	English Name	Scale Degrees
Major Scale / Ionian Mode	1, 2, 3, 4, 5, 6, 7	Natural Minor Scale / Aeolian Mode	1, 2, ♭3, 4, 5, ♭6, ♭7	Harmonic Minor Scale	1, 2, ♭3, 4, 5, ♭6, 7
Melodic Minor Scale	1, 2, ♭3, 4, 5, 6, 7	Dorian Mode	1, 2, ♭3, 4, 5, 6, ♭7	Phrygian Mode	1, ♭2, ♭3, 4, 5, ♭6, ♭7
Lydian Mode	1, 2, 3, ♯4, 5, 6, 7	Mixolydian Mode	1, 2, 3, 4, 5, 6, ♭7	Locrian Mode	1, ♭2, ♭3, 4, ♭5, ♭6, ♭7

5.2 Harmonic & Melodic Minor Derived Scales

English Name	Scale Degrees	English Name	Scale Degrees	English Name	Scale Degrees
Locrian ♮6 Scale	1, ♭2, ♭3, 4, ♭5, 6, ♭7	Ionian ♯5 Scale	1, 2, 3, 4, ♯5, 6, 7	Dorian ♯4 Scale	1, 2, ♭3, ♯4, 5, 6, ♭7
Phrygian Dominant (H.M. P5 Below)	1, ♭2, 3, 4, 5, ♭6, ♭7	Lydian ♯2 Scale	1, ♯2, 3, ♯4, 5, 6, 7	Super Locrian ♭7 Scale	1, ♭2, ♭3, ♭4, ♭5, ♭6, ♭♭7
Dorian ♭2 Scale	1, ♭2, ♭3, 4,	Lydian ♯5 Scale / Ly	1, 2, 3, ♯4, ♯5,	Lydian ♭7 Scale / Aco	1, 2, 3, ♯4, 5, 6,

English Name	Scale Degrees	English Name	Scale Degrees	English Name	Scale Degrees
	5, 6, ♭7	dian Augmented	6, 7	ustic Scale	♭7
Mixolydian ♭6 Scale	1, 2, 3, 4, 5, ♭6, ♭7	Locrian ♮2 Scale / Half-Diminished	1, 2, ♭3, 4, ♭5, ♭6, ♭7	Super Locrian Scale / Altered Scale	1, ♭2, ♭3, ♭4, ♭5, ♭6, ♭7

5.3 Symmetrical, Pentatonic & Special Scales

English Name	Scale Degrees	English Name	Scale Degrees	English Name	Scale Degrees
Whole Tone Scale	1, 2, 3, ♯4, ♯5, ♯6	Diminished Scale (Half-Whole)	1, ♭2, ♭3, 3, ♯4, 5, 6, ♭7	Diminished Scale (Whole-Half)	1, 2, ♭3, 4, ♭5, ♭6, 6, 7
Chromatic Scale	1, ♯1, 2, ♯2, 3, 4, ♯4, 5, ♯5, 6, ♯6, 7	Symmetrical Augmented Scale	1, ♭3, 3, 5, ♯5, 7	Major Pentatonic Scale	1, 2, 3, 5, 6
Minor Pentatonic Scale	1, ♭3, 4, 5, ♭7	Blues Scale	1, ♭3, 4, ♭5, 5, ♭7	Bebop Dominant Scale	1, 2, 3, 4, 5, 6, ♭7, 7
Prometheus Scale	1, 2, 3, ♯4, 6, ♭7	Enigmatic Scale	1, ♭2, 3, ♯4, ♯5, ♯6, 7	Tritone Scale	1, ♭2, 3, ♭5, 5, ♭7
Harmonic Major Scale	1, 2, 3, 4, 5, ♭6, 7	Major Locrian Scale	1, 2, 3, 4, ♭5, ♭6, ♭7	Neapolitan Major Scale	1, ♭2, ♭3, 4, 5, 6, 7

English Name	Scale Degrees	English Name	Scale Degrees	English Name	Scale Degrees
Neapolitan Minor Scale	1, ♭2, ♭3, 4, 5, ♭6, 7				

5.4 Japanese Traditional & Modern Scales

English Name	Scale Degrees	English Name	Scale Degrees	English Name	Scale Degrees
Ryosenpou	1, 2, 3, 5, 6	Ritsusenpou	1, 2, 4, 5, 6	Yousenpou / Inakabushi	1, 2, 4, 5, ♭7
Insenpou / Miyakobushi / Insen	1, ♭2, 4, 5, ♭7	Ryukyu Scale	1, 3, 4, 5, 7	Yona Nuki Major Scale	1, 2, 3, 5, 6
Yona Nuki Minor Scale	1, 2, ♭3, 5, ♭6	Niro Nuki Major Scale	1, ♭3, 4, 5, ♭7	Niro Nuki Minor Scale	1, 3, 4, 5, 7
Kumoi Scale	1, 2, ♭3, 5, 6	Iwato Scale	1, ♭2, 4, ♭5, ♭7	Hirajoshi Scale	1, 2, ♭3, 5, ♭6

5.5 World Ethnic Scales

English Name	Scale Degrees	English Name	Scale Degrees	English Name	Scale Degrees
Gong Mode (Chinese Scale)	1, 2, 3, 5, 6	Shang Mode (Chinese Scale)	1, 2, 4, 5, ♭7	Jiao Mode (Chinese Scale)	1, ♭3, 4, ♭6, ♭7

English Name	Scale Degrees	English Name	Scale Degrees	English Name	Scale Degrees
Zhi Mode (Chinese Scale)	1, 2, 4, 5, 6	Yu Mode (Chinese Scale)	1, ♭3, 4, 5, ♭7	Gypsy Scale / Hungarian Gypsy	1, ♭2, 3, 4, 5, ♭6, 7
Spanish Scale / Flamenco Scale	1, ♭2, 3, 4, 5, ♭6, ♭7	Hungarian Scale / Hungarian Minor	1, 2, ♭3, ♯4, 5, ♭6, 7	Scottish Scale	1, 2, 3, 5, 6
Pelog Scale	1, ♭2, ♭3, 5, ♭6	Bilaval Thaat	1, 2, 3, 4, 5, 6, 7	Khamaj Thaat	1, 2, 3, 4, 5, 6, ♭7
Kafi Thaat	1, 2, ♭3, 4, 5, 6, ♭7	Asavari Thaat	1, 2, ♭3, 4, 5, ♭6, ♭7	Bhairav Thaat	1, ♭2, 3, 4, 5, ♭6, 7
Bhairavi Thaat	1, 2, ♭3, 4, 5, ♭6, ♭7	Kalyan Thaat	1, 2, 3, ♯4, 5, 6, 7	Marwa Thaat	1, ♭2, 3, ♯4, 5, 6, 7
Poorvi Thaat	1, ♭2, 3, ♯4, 5, ♭6, 7	Todi Thaat	1, ♭2, ♭3, ♯4, 5, ♭6, 7	Maqam Rast	1, 2, ♭3 (Quarter Tone), 4, 5, 6, ♭7 (Quarter Tone)
Maqam Nahawand	1, 2, ♭3, 4, 5, ♭6, 7	Maqam Nawa Athar	1, 2, ♭3, ♯4, 5, ♭6, 7	Algerian Scale	1, 2, ♭3, ♯4, 5, ♭6, 7
Double Harmonic / Byzantine / Arabic	1, ♭2, 3, 4, 5, ♭6, 7				

6. Song Cadence Patterns

Refer to the following techniques when creating the ending section of a song.

- To create the feeling that the song is coming to an end, it is essential to establish a sense of finality. Consider using the following methods:
  - **Authentic Cadence**
    - Progression: Dominant (V or V_7) to Tonic (I)
    - Characteristics: Provides the strongest sense of stability. It functions like a period (.) at the end of a sentence, clearly signaling to the listener that the song has concluded.
  - **Half Cadence**
    - Progression: Tonic (or other chords) pausing on the Dominant (V)
    - Characteristics: While it creates a temporary pause, it leaves a floating, unresolved feeling. It acts like a comma (,) in a sentence, building anticipation for what comes next.
  - **Plagal Cadence**
    - Progression: Subdominant (IV) to Tonic (I)
    - Characteristics: Commonly known as the "Amen Cadence" used in church music, it delivers a gentle, soft, and soothing resolution.
  - **Deceptive Cadence**
    - Progression: Dominant (V) to a chord other than the Tonic (typically VI)
    - Characteristics: Just when the listener expects a resolution, it subverts expectations by moving to a different chord, creating a dramatic effect that suggests the music is still continuing.

Additionally, sustaining the final note for a longer duration strongly reinforces the feeling that the song has ended. Conversely, when creating loopable tracks, maintaining shorter notes until the end without following these cadence patterns ensures a smooth and natural transition when the song seamlessly loops back to the beginning.

7. Various Techniques for Song Composition

- To avoid creating a monotonous song, it is not recommended to have consecutive, adjacent notes of the same duration for more than four measures.
- In music for scenes such as battles, you can convey a sense of tension by making the melody’s pitch fluctuations more dramatic and increasing the tempo. Consider making

extensive use of sixteenth notes and thirty-second notes and a tempo of 130 BPM or higher.

- In music for horror scenes or dungeons, you can express tension and anxiety by deliberately incorporating a greater number of dissonant chords.
- It is not recommended to structure a piece entirely as multiple repetitions of the same pattern from start to finish, as this leads to monotony. Try to avoid monotony by, for example, repeating the same phrase twice, inserting a different phrase once, and then returning to the original phrase.
- For tracks set in towns or open fields, music with a sense of expansiveness is preferred. It's best to create a refreshing atmosphere by letting each note ring out a bit longer. In the case of field music, there tends to be a preference for pieces that sound like marches in 2/4 time.
- **MANDATORY** It is not acceptable for an entire track to consist solely of a single looped phrase across all channels. For example, a 1.5-minute song in which all channels repeat the same phrase three times is, in effect, only a 30-second song. Be sure to introduce variations in some part of the track—such as the melody or accompaniment—to avoid having the entire song consist of nothing but repetitions of the exact same phrase.
- When composing music that evokes a sense of vastness—such as the sky or the ocean—it is particularly important to vary the duration of the notes. Avoid pieces that consist solely of a continuous sequence of quarter notes. You can also create a sense of grandeur with a slow tempo, such as one at 100 BPM or below.
- When creating a mysterious track, using sustained sounds like `@52` Choir Aahs, `@53` Voice Oohs, `@50` Synth Strings, and `@98` Crystal with longer note durations can really set the mood.

---

8. Duration Calculation & Target Lengths

- **Calculation Formula**:
  - $¥text{Duration of 1 Beat (Quarter Note)} = ¥frac{60}{¥text{BPM}} ¥text{ seconds}$
  - At `T120`, a 4/4 bar (4 quarter notes) = **2.0 seconds**.
  - A 2-minute (120 seconds) song at `T120` requires **60 bars** of music.
- Use the loop macro `[ ... ]N` efficiently to build verse/chorus repetitions and achieve requested target durations accurately.

---

9. Output Format & Execution Role Rules (CRITICAL)

Single Code Block Constraint

ALWAYS output the ENTIRE MML content inside a **SINGLE code block**. Do NOT split channels into separate blocks or files. Write CH1, CH2, CH3, and CH4 consecutively in one unified text.

Boilerplate Example (Short BGM Template)

```
; ==========================================
; pisstrak Sample BGM - "Upbeat Town" (T128)
; ==========================================
; --- CH1: Lead Melody ---
CH1: T128 O5 V14 @0 L8
CH1: C D E F G4 E4 | A4 F4 G2 | F E D C < B4 > D4 | C2. R4
CH1: C E D F G4 E4 | F4 A4 G2 | D E F C < B4 > D4 | C2. R4
; --- CH2: Accompaniment Arpeggio ---
CH2: T128 O4 V10 @24 L8
CH2: [ C E G E C E G E | F A > C < A F A > C < A | G B > D < B G B > D < B | C
E G E C4 R4 ]2
; --- CH3: Bassline ---
CH3: T128 O3 V12 @33 L4
CH3: [ C R C R | F R F R | G R G R | C R C2 ]2
```

Rule A: Primary Output Contract (Chat AI Mode)

- **If you are acting as a Chat AI (Interactive Assistant):**
  - **ALWAYS output ONLY the raw MML text block.**
  - **DO NOT attempt to render, compile, generate, or attach audio files (`.wav` / `.ogg`).**
  - The user will take your generated MML text and feed it into `pisstrak` on their local machine. Outputting raw text is your 100% complete and final deliverable.

Rule B: Autonomous Pipeline Executions (Agentic AI Mode)

- **If you are acting as an Autonomous Agent (with direct CLI tool execution access):**
  - Generate the MML text content first.
  - Execute `pisstrak <input.mml> <output.ogg>` directly via CLI to render the audio file into the project asset directory.

10. Quick Start & Render Command Reference (For Assistant AI)

If the user asks how to render, play, or convert the generated MML file (e.g., "How do I render this?", "How to convert to WAV?"), provide the following `pisstrak` command-line instructions directly to the user.

```
pisstrak <input.mml> [output.wav|output.ogg] [tail_seconds] [soundfont.sf2]
```

- *Example:* `pisstrak mymusic.mml mymusic.ogg 2.5 mysoundfont.sf2`
- *Parameters:*
  - `<input.mml>`: Input MML score file.
  - `[output.wav|output.ogg]`: Output audio file path (Supports both `.wav` and `.ogg`).
  - `[tail_seconds]`: Extra fadeout/decay duration in seconds appended to the end of the rendered track (e.g., `2.5` adds 2.5s tail padding). If omitted (default `0`), the audio cuts off precisely at the final beat, which is ideal for seamless looping BGM tracks.
  - `[soundfont.sf2]`: SoundFont file for rendering (Defaults to `soundfont.sf2` if omitted).

SKILL_FB.md - fartbox Integration & Synthesis Talent Chip Version 0.11

1. Overview & Philosophy

Tool Identity

`fartbox` (Fast Audio Rendering Tool) is a 1-shot procedural sound effect generator designed for the TinCLE ecosystem. It takes text-based synth parameters (`.fart` files) or CLI arguments and renders uncompressed, high-fidelity 16-bit 44.1kHz WAV files in milliseconds.

---

2. CLI Command Specification

`fartbox` provides three core operational workflows via command-line flags:

Pattern A: Preset Generation & Parameter Export

Generates a sound based on a named preset algorithm, outputs the WAV file, and optionally saves the generated parameters to a `.fart` text file for future editing.

```
fartbox -t <preset_name> -o <output.wav> [-save <output.fart>] [-s <seed_number>]
```

- `-t <preset_name>`: Specifies the sound preset generator.
- `-o <output.wav>`: Output target WAV file path.
- `-save <output.fart>`: (Optional) Saves the randomized parameters of the generated preset to a text file.
- `-s <seed_number>`: (Optional) Integer seed for reproducible pseudo-random generation.

Pattern B: Direct Synthesis from Parameter File (`.fart`)

Reads an existing `.fart` parameter text file and renders it directly into a WAV file. This is the primary method for AI agents to write custom audio parameters.

```
fartbox -c <input.fart> -o <output.wav>
```

- `-c <input.fart>`: Source `.fart` parameter configuration file.
- `-o <output.wav>`: Output target WAV file path.

Pattern C: Mutation Workflow

Loads a `.fart` parameter file, applies slight random variations (mutations) to the active parameters, renders the mutated WAV, and optionally saves the new parameters.

```
fartbox -c <input.fart> -m -o <output.wav> [-save <mutated.fart>] [-s <seed_number>]
```

- `-c <input.fart>`: Base parameter file to mutate.
- `-m`: Enables mutation mode.
- `-o <output.wav>`: Output target WAV file path for mutated sound.
- `-save <mutated.fart>`: (Optional) Saves the newly mutated parameters.
- `-s <seed_number>`: (Optional) Seed controlling the mutation variance.

---

3. Preset Types Reference

The following preset identifiers can be passed to the `-t` option:

- `coin` / `pickup`: Short, high-pitched arpeggiated tones suited for item collection or scoring.
- `laser` / `shoot`: Frequency-sweeping downward tones for projectiles and energy weapons.
- `explosion`: Noise-based low-frequency rumble with pitch drop and distortion.
- `powerup`: Rising pitch sweeps with vibrato or arpeggiation for enhancements.
- `hit` / `hurt`: Short, impactful noise or square waves for damage feedback.
- `jump`: Upward frequency slides for character locomotion.
- `blip` / `select`: Extremely short, high-frequency tones for UI interaction.
- `random`: Unconstrained parameter randomization for abstract or unusual sound effects.

4. `.fart` Parameter Reference

The `.fart` parameter file is a plain-text key-value configuration file containing all synthesis parameters required by the `fartbox` synth core. Each line consists of `parameter_name = float_value`. Comments starting with `#` are ignored.

All parameters accept floating-point numbers. The valid parameter keys and their ranges/effects are detailed below:

Waveform Selection

- `wave_type` (Integer: 0, 1, 2, 3)
  - `0`: Square / Pulse wave (classic retro 8-bit chip synth).
  - `1`: Sawtooth wave (harsh, rich harmonic spectrum).
  - `2`: Sine wave (pure fundamental tone, smooth).
  - `3`: White Noise wave (percussive sounds, explosions, impacts).

Pitch & Frequency

- `p_base_freq` (Float: 0.0 to 1.0)
  - Primary synthesis frequency (pitch). Higher values yield higher tones.
- `p_freq_limit` (Float: 0.0 to 1.0)
  - Lower cutoff limit for frequency sweeps. If a downward frequency sweep hits this limit, generation terminates.
- `p_freq_ramp` (Float: -1.0 to 1.0)
  - Frequency slide rate. Negative values slide pitch down (lasers, hits); positive values slide pitch up (jumps, powerups).
- `p_freq_dramp` (Float: -1.0 to 1.0)
  - Acceleration of the frequency slide. Controls non-linear pitch curving.

Pulse Width / Duty Cycle (Square Wave Only)

- `p_duty` (Float: 0.0 to 1.0)
  - Pulse width modulation for square waves (`wave_type = 0`). Controls timbre thickness.
- `p_duty_ramp` (Float: -1.0 to 1.0)
  - Sweeps the pulse width over time, introducing dynamic harmonic motion.

Pitch Vibrato

- `p_vib_strength` (Float: 0.0 to 1.0)
  - Pitch modulation depth (vibrato intensity).

- `p_vib_speed` (Float: 0.0 to 1.0)
  - Modulation frequency / speed of vibrato.
- `p_vib_delay` (Float: 0.0 to 1.0)
  - Delay before vibrato modulation engages.

ADSR Envelope

- `p_env_attack` (Float: 0.0 to 1.0)
  - Attack time. 0.0 provides instant impact; higher values create a fade-in.
- `p_env_sustain` (Float: 0.0 to 1.0)
  - Hold time at peak amplitude.
- `p_env_decay` (Float: 0.0 to 1.0)
  - Release / decay duration. Controls overall sound length.
- `p_env_punch` (Float: 0.0 to 1.0)
  - Initial volume boost during attack phase, adding punch to impacts.

Low-Pass Filter (LPF)

- `p_lpf_freq` (Float: 0.0 to 1.0)
  - LPF cutoff frequency. 1.0 disables filter; lower values muffle the sound.
- `p_lpf_ramp` (Float: -1.0 to 1.0)
  - Cutoff frequency sweep over time.
- `p_lpf_resonance` (Float: 0.0 to 1.0)
  - Resonance multiplier near the cutoff frequency, adding squelch or metallic ringing.

High-Pass Filter (HPF)

- `p_hpf_freq` (Float: 0.0 to 1.0)
  - HPF cutoff frequency. Removes bass frequencies. Higher values result in thinner sounds.
- `p_hpf_ramp` (Float: -1.0 to 1.0)
  - HPF Cutoff sweep over time.

Phaser Effect

- `p_pha_offset` (Float: -1.0 to 1.0)
  - Phase cancellation offset, creating jet-engine or swooshing effects.
- `p_pha_ramp` (Float: -1.0 to 1.0)
  - Phase shift speed and direction.

Arpeggio & Repeat

- `p_repeat_speed` (Float: 0.0 to 1.0)
  - Triggers automatic re-attatck of the sound at specified intervals (stutter effect).
- `p_arp_speed` (Float: 0.0 to 1.0)
  - Speed of two-note pitch interval jumping.
- `p_arp_mod` (Float: -1.0 to 1.0)
  - Pitch interval for arpeggiation. Positive jumps pitch up (e.g. coin effect); negative drops pitch.

---

5. AI Workflow Guidelines

When an AI agent needs to generate sound assets for a TinCLE project, it must adhere to the following decision tree:

Step 1: Initial Prototype Phase

Always start by generating a baseline audio asset using a standard preset. Save the text configuration (`.fart`) alongside the WAV file so parameters can be inspected or modified later.

```
fartbox -t laser -o assets/sfx/laser.wav -save assets/sfx/laser.fart
```

Step 2: Variation Generation (Mutation)

If multiple distinct sound variations are required (e.g., subtle variations of footstep or explosion sounds), load the base `.fart` file and execute mutation mode.

```
fartbox -c assets/sfx/explosion_base.fart -m -o assets/sfx/explosion_var01.wav
-save assets/sfx/explosion_var01.fart -s 101
fartbox -c assets/sfx/explosion_base.fart -m -o assets/sfx/explosion_var02.wav
-save assets/sfx/explosion_var02.fart -s 102
```

Step 3: Precision Manual Tuning

When specific sound design constraints are requested (e.g., "Make the laser decay longer" or "Make the explosion lower pitched"):

1. Do NOT re-run randomized presets.
2. Open and inspect the existing `.fart` parameter file.

3. Modify specific parameters directly:
    - To lengthen decay: Increase `p_env_decay`.
    - To lower pitch: Decrease `p_base_freq`.
    - To add crunch/punch: Increase `p_env_punch` or adjust `p_lpf_resonance`.
4. Re-render the audio directly from the edited `.fart` file:

```
fartbox -c assets/sfx/laser.fart -o assets/sfx/laser.wav
```

Rule A: Primary Output Contract (Chat AI Mode)

- **If you are acting as a Chat AI (Interactive Assistant without direct CLI access):**
    - **ALWAYS output ONLY the raw `.fart` text parameter block.**
    - **DO NOT attempt to render, compile, generate, or attach audio files (`.wav`).**
    - The frontend client environment (e.g., mobile chat apps) may fail to display or send rendered binary audio attachments, causing complete response silences.
    - The user will take your generated `.fart` text configuration and feed it directly into `fartbox` on their local machine. Outputting clean, copyable text parameters is your 100% complete and final deliverable.

Rule B: Autonomous Pipeline Executions (Agentic AI Mode)

- **If you are acting as an Autonomous Agent (with direct CLI execution access):**
    - Generate or edit the `.fart` parameter text file first.
    - Execute `fartbox -c <input.fart> -o <output.wav>` directly via CLI to render the audio asset into the project's sound directory.

6. `.fart` Boilerplate Templates

Below are standard boilerplate configurations for common sound categories. AI agents should use these templates as starting baselines for direct editing or generation.

Template 1: Standard Laser / Projectile (`laser_base.fart`)

```
# fartbox Sound Parameter - Laser Base
wave_type = 0
p_base_freq = 0.600000
p_freq_limit = 0.200000
p_freq_ramp = -0.250000
p_freq_dramp = 0.000000
p_duty = 0.400000
```

```
p_duty_ramp = -0.100000
p_vib_strength = 0.000000
p_vib_speed = 0.000000
p_vib_delay = 0.000000
p_env_attack = 0.000000
p_env_sustain = 0.100000
p_env_decay = 0.200000
p_env_punch = 0.100000
p_lpf_resonance = 0.000000
p_lpf_freq = 1.000000
p_lpf_ramp = 0.000000
p_hpf_freq = 0.100000
p_hpf_ramp = 0.000000
p_pha_offset = 0.000000
p_pha_ramp = 0.000000
p_repeat_speed = 0.000000
p_arp_speed = 0.000000
p_arp_mod = 0.000000
```

Template 2: Heavy Explosion / Impact (`explosion_base.fart`)

```
# fartbox Sound Parameter - Explosion Base
wave_type = 3
p_base_freq = 0.250000
p_freq_limit = 0.000000
p_freq_ramp = -0.150000
p_freq_dramp = 0.000000
p_duty = 0.000000
p_duty_ramp = 0.000000
p_vib_strength = 0.200000
p_vib_speed = 0.300000
p_vib_delay = 0.000000
p_env_attack = 0.000000
p_env_sustain = 0.200000
p_env_decay = 0.500000
p_env_punch = 0.400000
p_lpf_resonance = 0.300000
p_lpf_freq = 0.800000
p_lpf_ramp = -0.100000
p_hpf_freq = 0.000000
p_hpf_ramp = 0.000000
p_pha_offset = -0.100000
p_pha_ramp = -0.100000
p_repeat_speed = 0.000000
p_arp_speed = 0.000000
```

```
p_arp_mod = 0.000000
```

Template 3: 8-Bit Coin / Item Pickup (`coin_base.fart`)

```
# fartbox Sound Parameter - Coin Base
wave_type = 0
p_base_freq = 0.450000
p_freq_limit = 0.000000
p_freq_ramp = 0.000000
p_freq_dramp = 0.000000
p_duty = 0.500000
p_duty_ramp = 0.000000
p_vib_strength = 0.000000
p_vib_speed = 0.000000
p_vib_delay = 0.000000
p_env_attack = 0.000000
p_env_sustain = 0.080000
p_env_decay = 0.250000
p_env_punch = 0.300000
p_lpf_resonance = 0.000000
p_lpf_freq = 1.000000
p_lpf_ramp = 0.000000
p_hpf_freq = 0.000000
p_hpf_ramp = 0.000000
p_pha_offset = 0.000000
p_pha_ramp = 0.000000
p_repeat_speed = 0.000000
p_arp_speed = 0.600000
p_arp_mod = 0.500000
```

Template 4: UI Cursor / Select Blip (`blip_base.fart`)

```
# fartbox Sound Parameter - UI Blip Base
wave_type = 0
p_base_freq = 0.350000
p_freq_limit = 0.000000
p_freq_ramp = 0.000000
p_freq_dramp = 0.000000
p_duty = 0.500000
p_duty_ramp = 0.000000
p_vib_strength = 0.000000
p_vib_speed = 0.000000
p_vib_delay = 0.000000
p_env_attack = 0.000000
```

```
p_env_sustain = 0.050000
p_env_decay = 0.100000
p_env_punch = 0.000000
p_lpf_resonance = 0.000000
p_lpf_freq = 1.000000
p_lpf_ramp = 0.000000
p_hpf_freq = 0.150000
p_hpf_ramp = 0.000000
p_pha_offset = 0.000000
p_pha_ramp = 0.000000
p_repeat_speed = 0.000000
p_arp_speed = 0.000000
p_arp_mod = 0.000000
```

TinCLE Version 0.11 Talent Chip: pubicnet (SKILL_PN.md)

~ Multi-User Dedicated Server Generator & Asynchronous Network API ~

1. Overview & Architecture

The TinCLE networking pipeline consists of three core components:

1. **`pubicnet` (CLI Tool)**
   - Generates/reads `uuid.txt` (36-byte UUID) in the project directory.
   - Binary-patches the template dynamic binary with the 36-character GAME_ID (UUID) and Game Title.
   - Outputs a standalone dedicated server executable: `my_server.exe`.
2. **`my_server.exe` (Dedicated Server)**
   - Listens on TCP port `8080` (default) for non-blocking packet broadcasting.
   - Validates client header MAGIC (`"PNET"`) and GAME_ID (UUID).
   - Automatically assigns unique `peer_id` (1~65535) to connected clients upon handshake.
   - Broadcasts incoming payload data to all clients residing in the same `room_id`.
3. **`tincle_net_*` API (Client Integration via `tincle.dll`)**
   - Non-blocking socket communication layer embedded inside TinCLE.
   - Cross-platform support (Win32 / WASM via Emscripten).

---

2. Server Generation Workflow (Step-by-Step)

Before invoking network APIs in client code (`game.c`), the AI Agent or developer must generate the dedicated game server executable.

Step 1: Execute `pubicnet`

Run the `pubicnet` tool from the project directory with the game title as a required argument. You may optionally specify a custom output executable name (defaults to `my_server.exe` if omitted):

```
# Default output (my_server.exe)
pubicnet "My Game Title"
# Custom output name
pubicnet "My Game Title" custom_server.exe
```

Step 2: Automatic UUID & Server Output

- If `uuid.txt` does NOT exist in the working directory, `pubicnet` generates a new UUID (e.g., `a1b2c3d4-5678-90ab-cdef-1234567890ab`) and saves it as `uuid.txt`.
- `pubicnet` reads the template library (`pubicnet.dll`), injects the target full 36-character GAME_ID (UUID) and Game Title into internal memory placeholders, and outputs the patched server binary (`my_server.exe` or specified name).

Step 3: Launch Dedicated Server

Run `my_server.exe` to start listening for client connections:

```
my_server -p 8080
```

Console Output:

```
================================================================================
 [TinCLE Version 0.11] My Game Title Server Started!
 [Port] 8080
================================================================================
```

Step 4: Server Packaging & Release Preparation

The generated server executable is unique to your game build and **MUST be distributed alongside the game client**.

To ensure safety, stability, and ease of use, follow these distribution guidelines:

1. **Apply `poopee` Post-Processing (CRITICAL):**
    - Before distributing the server executable, you **MUST** run `poopee` on the generated server EXE to embed the application icon and OS manifest.
    - This step is strongly recommended to prevent false-positive antivirus flags (such as Windows Defender) on user machines.

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

2. **Standalone Server ZIP Distribution:**
    - For maximum user convenience, create a separate standalone ZIP archive specifically for host players/server admins (e.g., `MyGame_Dedicated_Server.zip`).
    - The archive should contain the following 3 files:
        1. `my_server.exe` (The `poopee`-processed server executable)
        2. `CREDITS.txt` (License and attribution credits)
        3. `readme.txt` (Clear documentation explaining what the server is, how to open ports, and how to launch it)

---

3. Client API Reference

All networking functions are exposed via `libs/tincle.h`.

Function	Return Type	Description
`tincle_net_set_game_id(const char* game_id_or_uuid)`	`void`	Configures full 36-character GAME_ID (UUID) used for server authentication. **Must be called before `tincle_net_connect`**.
`tincle_net_connect(const char* ip, int port, uint16_t room_id)`	`int`	Initiates asynchronous TCP connection to the server and triggers automatic Peer ID handshake. Returns `1` on socket creation success, `0` on failure.
`tincle_net_disconnect()`	`void`	Closes network socket and resets client Peer ID state.
`tincle_net_send(const uint8_t* data, uint16_t size)`	`int`	Broadcasts raw binary payload to all clients in the current `room_id`. Returns `1` on success, `0` on failure.

Function	Return Type	Description
`tincle_net_poll(uint16_t* out_sender_peer_id, uint8_t* out_buffer, uint16_t max_size)`	`int`	Non-blocking receive function. Must be called repeatedly in the game update loop. Returns received payload byte count (`0` if no data). **If `out_sender_peer_id` is `0`, `out_buffer` contains the `uint16_t` Peer ID of a disconnected client sent by the server.**
`tincle_net_get_peer_id()`	`uint16_t`	Returns the assigned unique Client Peer ID (`1` ~ `65535`). Returns `0` if not yet connected/handshaken.

Disconnect & Timeout Handling Mechanism

To maintain a responsive multiplayer state, clients must handle disconnection using a two-stage approach inside the `tincle_net_poll` loop:

1. **Visual Hide Timeout (Local Client Side - 2 Seconds):**
    - If a peer's position data has not arrived for more than 2 seconds (`PEER_TIMEOUT_SEC`), the client temporarily hides the player from the screen (e.g., `active = false`). This prevents dead/lagging players from lingering on the screen.
2. **Server Logout Notification (System Event - 0s or 30s):**
    - When `tincle_net_poll` returns `out_sender_peer_id == 0` , it indicates a system event directly sent from the server.
    - The payload contains a `uint16_t` representing the `disconnected_peer_id` .
    - **Immediate Disconnect (0s):** Sent instantly when a client performs a graceful close/disconnect.
    - **Timeout Disconnect (30s):** Sent after 30 seconds if a client suffers a hard crash or network disruption without sending a WebSocket close frame.
    - Upon receiving `out_sender_peer_id == 0` , the client must completely clean up and release the specified peer's state, regardless of whether the player was already visually hidden by the 2-second local check.

4. Minimal Network Implementation Boilerplate

Critical Architecture Rules for AI Agents

When designing a TinCLE multiplayer game, AI Agents **MUST** strictly adhere to the following architecture rules:

1. **Do NOT Rely on `uuid.txt` in the Distributed Game:**
    - The `uuid.txt` file is **ONLY** used locally during the server generation build process with `pubicnet`.
    - Never assume `uuid.txt` will exist on the player's machine. The client code (`game.c`) **MUST** embed or hardcode the full 36-character GAME_ID (UUID) directly in C variables and pass it to `tincle_net_set_game_id()`.
2. **Always Provide a Server IP / Port Configuration UI:**
    - **NEVER** hardcode the destination IP address to `127.0.0.1` (localhost) or a fixed IP inside the client code.
    - Hardcoding `127.0.0.1` completely breaks real online multiplayer across different machines.
    - You **MUST** implement an in-game UI screen (or input prompt/config file loader) that allows players to freely enter and save the target Server IP address and Port number before attempting to call `tincle_net_connect()`.
3. **UUID Acquisition Rule by AI Agent Type (CRITICAL):**
    - The 36-character GAME_ID (UUID) passed to `tincle_net_set_game_id()` MUST strictly match the 36-characters UUID embedded inside the generated server (`my_server.exe`). Never invent or hallucinate a random dummy UUID!
    - **For Autonomous / Agentic AIs (Tools & CLI execution enabled):** You MUST execute `pubicnet "<Game Title>"` FIRST in the terminal to generate `uuid.txt` and `my_server.exe`. Then, read the generated UUID from `uuid.txt` and embed the full 36-characters directly into the client C code (`game.c`).
    - **For Conversational / Chat AIs (No direct terminal execution):** You CANNOT run `pubicnet` by yourself. If the user has NOT provided the target UUID or `uuid.txt` content yet, you **MUST explicitly ask the user to provide the UUID** (or run `pubicnet` first) BEFORE writing the `tincle_net_set_game_id()` code. NEVER invent a placeholder UUID, as mismatched GAME_IDs will permanently block all network connections!

Boilerplate

This boilerplate demonstrates how an AI Agent should write client code (`game.c`) within the official TinCLE state lifecycle to establish connections, send player state, and poll incoming broadcast data without blocking the render thread.

CRITICAL Note: Always call `tincle_net_set_game_id()` before `tincle_net_connect()` to ensure the full 36-character GAME_ID (UUID) matches the server.

```
#define SDL_MAIN_HANDLED
#include "libs/tincle.h"
#include <stdio.h>
#include <stdint.h>
#ifdef __EMSCRIPTEN__
    #include <SDL2/SDL_scancode.h>
#else
    #include "libs/SDL_scancode.h"
    #include <windows.h>
#endif
// Network Packet Structure
typedef struct {
    float x;
    float y;
} PlayerNetworkPacket;
// Global States & Storage
static int g_font_medium = -1;
static float g_my_x = 320.0f;
static float g_my_y = 240.0f;
// Table for other players' positions (Supports up to 16 peers)
static float g_other_x[16] = {0};
static float g_other_y[16] = {0};
static bool  g_other_active[16] = {false};
static float g_other_last_seen[16] = {0};
// Timeout threshold for peer disconnections (seconds)
#define PEER_TIMEOUT_SEC 2.0f
// State 0: Gameplay Logic Updates
void play_update(float dt) {
    static float s_total_time = 0.0f;
    s_total_time += dt;
    // 1. Network Receive Polling (Process all incoming packets)
    uint16_t sender_peer_id = 0;
    uint8_t recv_buf[128]; // Receive via buffer to also catch system notificat
ions (leave events)
    int recv_len = 0;
    // 1. Network Receive Polling
    while ((recv_len = tincle_net_poll(&sender_peer_id, recv_buf, sizeof(recv_b
uf))) > 0) {
        // sender_peer_id == 0 indicates a system notification from the server
(immediate disconnect or 30-sec timeout disconnect)
```

```
        if (sender_peer_id == 0) {
            if (recv_len >= (int)sizeof(uint16_t)) {
                uint16_t left_peer_id = *(uint16_t*)recv_buf;
                if (left_peer_id > 0 && left_peer_id < 16) {
                    // Output a "completely logged out" log, regardless of whet
her the player was already visually hidden by the 2-sec timeout!
                    printf("[Net] Peer %d completely logged out (Server Notific
ation)!\n", left_peer_id);
                    g_other_active[left_peer_id] = false; // Completely release
and reset the player state
                }
            }
        }
        // Normal position synchronization packets from other players
        else if (sender_peer_id != tincle_net_get_peer_id() && sender_peer_id <
16) {
            if (recv_len >= (int)sizeof(PlayerNetworkPacket)) {
                PlayerNetworkPacket* recv_packet = (PlayerNetworkPacket*)recv_b
uf;
                g_other_x[sender_peer_id] = recv_packet->x;
                g_other_y[sender_peer_id] = recv_packet->y;
                g_other_active[sender_peer_id] = true;
                g_other_last_seen[sender_peer_id] = s_total_time;
            }
        }
    }
    // 2. Visual Timeout Check (Emergency visual hide processing)
    for (int i = 1; i < 16; i++) {
        if (i != tincle_net_get_peer_id() && g_other_active[i]) {
            // Temporarily hide the player from the screen if packets stop arri
ving for 2 seconds
            if (s_total_time - g_other_last_seen[i] > PEER_TIMEOUT_SEC) {
                printf("[Net] Peer %d position data paused (Visual Hide)\n",
i);
                g_other_active[i] = false;
            }
        }
    }
    // 3. Local Player Input Handling
    float move_speed = 200.0f;
    if (tincle_key_pressed(SDL_SCANCODE_LEFT))  g_my_x -= move_speed * dt;
    if (tincle_key_pressed(SDL_SCANCODE_RIGHT)) g_my_x += move_speed * dt;
    if (tincle_key_pressed(SDL_SCANCODE_UP))    g_my_y -= move_speed * dt;
    if (tincle_key_pressed(SDL_SCANCODE_DOWN))  g_my_y += move_speed * dt;
    // 4. Send Local Position to Server (Only after Peer ID is assigned)
    uint16_t my_id = tincle_net_get_peer_id();
    if (my_id != 0) {
        PlayerNetworkPacket my_packet;
```

```
        my_packet.x = g_my_x;
        my_packet.y = g_my_y;
        tincle_net_send((const uint8_t*)&my_packet, sizeof(PlayerNetworkPacke
t));
    }
}
// State 0: Gameplay Rendering
void play_draw(void) {
    // Clear background (Dark Navy)
    tincle_clear((TinCLE_Color){20, 20, 50, 255});
    if (g_font_medium != -1) {
        tincle_set_font(g_font_medium);
    }
    uint16_t my_id = tincle_net_get_peer_id();
    char buf[64];
    // Render assigned Peer ID
    sprintf(buf, "MY PEER ID: %d", my_id);
    tincle_draw_text(buf, 20, 20, (TinCLE_Color){0, 255, 0, 255});
    // Render Local Player (Yellow)
    sprintf(buf, "P%d", my_id);
    tincle_draw_text_center(buf, (int)g_my_x, (int)g_my_y, (TinCLE_Color){255,
255, 0, 255});
    // Render Remote Peers (Magenta)
    for (int i = 1; i < 16; i++) {
        if (i != my_id && g_other_active[i]) {
            sprintf(buf, "P%d", i);
            tincle_draw_text_center(buf, (int)g_other_x[i], (int)g_other_y[i],
(TinCLE_Color){255, 0, 255, 255});
        }
    }
}
// Engine & Network Initialization
void game_load(void) {
    // Load default font asset
    g_font_medium = tincle_load_font("font.ttf", 24);
    // Set full 36-character Game UUID (Must match server config)
    tincle_net_set_game_id("406230b1-ae94-05b7-e9af-541d23f313c7");
    // Establish connection to pubicnet server (IP, Port, RoomID)
    if (tincle_net_connect("127.0.0.1", 8080, 1)) { // Server IP Address & Port
        printf("[Net] Connecting to pubicnet server...\n");
    }
    tincle_add_state(0, play_update, play_draw);
    tincle_change_state(0);
}
// Application Entry Point
#ifdef __EMSCRIPTEN__
int main(int argc, char* argv[]) {
```

```
    tincle_start(game_load, "My Net Game", true, 640, 480, TINCLE_MODE_2D_NORMA
L);
    return 0;
}
#else
int WINAPI WinMain(HINSTANCE h, HINSTANCE hp, LPSTR c, int n) {
    tincle_start(game_load, "My Net Game", true, 640, 480, TINCLE_MODE_2D_NORMA
L);
    return 0;
}
#endif
```