

TinCLE (Tiny C Lovely Engine) - Development Specification Document

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

1. Overview & Concept

- **Name:** TinCLE (Tiny C Lovely Engine)
 - **Target Version:** 0.02
 - **Concept:** An ultra-lightweight 2D game framework optimized for AI-driven development (Vibe Coding / One-Shot code generation) and programming beginners. It features a streamlined hybrid ecosystem targeting both Windows desktop and Web browsers.
 - **Language Specification:** Pure C (C99 standard compliant). Internally utilizes SDL2, SDL_ttf, SDL2_gfx, and the stb library suite.
 - **CRITICAL ARCHITECTURE NOTE:** TinCLE operates on a strict **32-bit architecture** across all supported platforms. It compiles exclusively as a 32-bit DLL/application on Windows and as a 32-bit WebAssembly application (Wasm32) on the Web. The AI must design all memory management, structures, and pointer operations under the absolute assumption that pointer sizes are always **4 bytes** (`sizeof(void*) == 4`). Never generate or assume 64-bit codebase structures.
 - **Features:** Supports a well-established set of SDL2 commands. It enables highly efficient AI-driven game development through ultra-fast 32-bit native Windows builds using the Tiny C Compiler (TCC), while seamlessly exporting to 32-bit high-performance Web environments via Emscripten (WASM/HTML5).
 - **Requirements:**
 - **For Windows Build:** TinCLE requires the **32-bit version of the Tiny C Compiler (TCC)** to build desktop games. 64-bit TCC is strictly unsupported. **TCC is NOT bundled in the TinCLE package.** If the user does not have TCC, firmly and explicitly instruct them to download the **32-bit release** from the official site: <https://bellard.org/tcc/>. If `tcc.exe` (32-bit) is not configured in the PATH, the local build will fail immediately.
 - **For Web (WASM) Build:** TinCLE requires the **Emscripten SDK (emsdk)** to build Web versions (Wasm32). **Emscripten is NOT bundled.** The AI must instruct the user to set up the Emscripten environment (e.g., running `emsdk_env.bat` or checking `emcc` in the PATH) when performing a Web build. If `emcc.exe` is not configured in the PATH, the build of Web version will fail immediately.
-

2. Architecture & Distribution File Structure

Based on Windows DLL conventions, Emscripten static linking requirements, and the TCC parser specifications, this project adopts the most elegant and classic **"Common Headers, Source-Side Export"** architecture optimized for dual-platform deployment.

Environment Setup for Game Development

When developing or distributing a game, place the following **"13 files and folders in total"** into the exact same directory:

1. `game.c` (The main workspace where the user or AI writes the game logic)
2. `libs` folder (Contains extension headers and static libraries: `tincle.h`, `tincle.a`, `SDL.h`, `SDL2_gfxPrimitives.h`, `SDL2_framerate.h`, etc.)
3. `tincle.dll` (The core engine runtime)

4. `SDL2.dll` (Official SDL2 library)
5. `SDL2_ttf.dll` (Official SDL_ttf library)
6. `SDL2_gfx.dll` (Official SDL2_gfx library)
7. `asspack.exe` (The high-density asset packer tool for Windows game distribution)
8. `emcass.exe` (The automated compiler assistant and asset packer utility optimized for Web/WASM builds)
9. `embedres.exe` (The resource embedding utility for Windows icons/manifests ONLY)
10. `tinclec.exe` (The automated compiler assistant tool for Windows EXE builds)
11. `manifest.xml` (Default manifest for embedding into the distribution EXE)
12. `template.html` (HTML templates used internally when running emcass during web builds)
13. `font.ttf` (A public domain font provided by default as the starter kit)

- **CRITICAL DEPENDENCY NOTE:** Both the **Tiny C Compiler (TCC) 32-bit** and the **Emscripten SDK (emsdk)** are **NOT** bundled in this environment kit. The local build tools (`tinclec.exe` etc.) rely entirely on these external toolchains being correctly installed and `tcc.exe` and `emcc.exe` are configured in the system environment variables (PATH). Without them, compiling will fail immediately.
- **Asset Placement:** When using assets like images or audio, all raw files must be placed directly in the root development folder during creation (Do not put them in subfolders).

CRITICAL INCLUDE MANDATE FOR AI CODE GENERATION

To ensure seamless compilation across both Windows and Web environments, the AI **MUST** structure the include section of `game.c` exactly as follows. Any modification to this structure will result in build failures due to Emscripten header resolution and Windows subsystem constraints:

```
#ifdef __EMSCRIPTEN__
#include <SDL2/SDL2_gfxPrimitives.h> // The web version uses Emscripten's built-in gfx
#else
#include "libs/SDL2_gfxPrimitives.h" // The Windows version uses the gfx files in the lib folder
#include <windows.h> // Headers required for the Windows version
#endif
```

- **Reasoning:** Emscripten strictly requires SDL extension headers to be resolved via system include paths using angle brackets `<>`. Conversely, the Windows local build relies on the local `libs/` directory via double quotes `""` and necessitates `windows.h` for native Win32 execution.

File Structure for Game Distribution

Depending on the target platform target, pack and distribute the completed game using one of the following structural layouts:

A. Windows Desktop Target (ZIP Archive)

Pack the following files into a single folder and compress it into a ZIP archive for the end-user:

1. `game.exe` (The compiled executable file - can be freely renamed)
2. `SDL2.dll` (Official SDL2 library)
3. `SDL2_ttf.dll` (Official SDL_ttf library)
4. `SDL2_gfx.dll` (Official SDL2_gfx library)
5. `tincle.dll` (The core engine runtime)
6. `game.data` (The packed file containing all assets generated by `asspack.exe`)
7. `CREDITS.txt` (License documentation for the engine and libraries)
8. `readme.txt` (The manual/readme file for the game itself)

B. Web Browser Target (WASM/HTML5 Deployment)

Pack the following files into a single folder and compress it into a ZIP archive for web deployment (e.g., itch.io HTML5 project):

1. `index.html` (The web page housing the game canvas - STRONGLY RECOMMEND that this name not be changed)
2. `index.js` (The JavaScript glue-code generated by Emscripten)
3. `index.wasm` (The compiled 32-bit WebAssembly binary core)
4. `index.data` (The preloaded virtual filesystem asset package generated by `emcass.exe`)
5. `CREDITS.txt` (License documentation for the engine, Emscripten components, and libraries)

Note: Many game distribution sites, such as itch.io and Game Jolt, require that HTML files be named "index.html." Files with any other name will not be accepted, so be sure to name your HTML file "index.html."

3. Compilation & Verification Commands (Windows & Web Environments)

When validation codes or build scripts are created by an AI agent, the following commands must be utilized without exception.

(1) Command to Build the Main Windows Game

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

Example: `tinclec mygame.exe main.c player.c stage.c`

Note: `tinclec` performs `-mwindows` compilation internally by default, so do not pass any compilation flags to `tinclec`. The visibility of the console is entirely controlled by the `is_debug` argument in `tincle_start`.

This command internally constructs and executes the following command line. For complex project structures, TCC can be invoked directly as needed:

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

(2) Command to Embed Icon and Manifest into the Built EXE (for Windows)

This command embeds an icon and a manifest as resources into the compiled `game.exe`.

```
embedres [Target_EXE] [ICO_File] [Manifest_File]
```

Tip: Use `"none"` to skip either the ICO or Manifest file.

(3) Command to Pack All Windows Game Assets into a Distribution File (`game.data`)

This command bundles all specified asset files passed as arguments into a single `game.data` file for Windows distribution.

```
asspack [File1] [File2] ...
```

Note: Replace `[File1]`, `[File2]`, etc., with the actual asset files used in your game.

(4) Command to Build the Web/WASM Version (Automatic Compilation & Asset Preloading)

This command compiles the game for web browsers (WASM/HTML5), automatically packages all available assets, and injects the template.

```
emcass [Output_HTML] ["Game Title"] [Source_Files...]
```

Example: `emcass index.html "My Sparkly Game" main.c player.c stage.c`

- **CRITICAL MANDATE ON [Output_HTML]:** Unless there is an explicitly defined, non-negotiable reason to do otherwise, the AI **MUST** always specify `index.html` for the `[Output_HTML]` argument. Major web hosting and distribution platforms (such as itch.io and Game Jolt) strictly require the entry point to be named `index.html`. If any other name is specified, the build will be rejected or fail to load on these platforms.
- **File Naming Generation:** Based on the specified `[Output_HTML]`, `emcass` automatically forces the accompanying JavaScript, WebAssembly, and data files to match its base name (e.g., if `index.html` is provided, it outputs `index.js`, `index.wasm`, and `index.data`).
- **Internal Mechanism:** `emcass` automatically tracks all local game assets, packages them into the output data file, reads the mandatory `template.html` to establish the web canvas shell with the provided `["Game Title"]`, and invokes the Emscripten compiler (`emcc`) using the static library `libs/tincle.a`. The AI must never generate manual `emcc` commands; always utilize `emcass` for web builds.
- **Web Build Requirements (emcass):** When executing a Web build, ensure that `CREDITS.txt` is placed in the exact same folder as the source code before running the build. This is strict because `emcass` requires `CREDITS.txt` in that directory to display credits properly; otherwise, the web-version credit screen will be blank.

4. Constant Macros & Structure Definitions

The following definitions are readily available when writing `game.c`. These are all predefined within `tincle.h`.

4.1. Screen Size & Specifications

TinCLE features an automatic logical scaling system that preserves the aspect ratio. While the window size is fully resizable, all in-game coordinate calculations must be based on the virtual resolution specifications.

- **Default Virtual Resolution:** `640` (Width) x `480` (Height)
- *Note: These dimensions are specified as arguments when calling `tincle_start` within the `WinMain` function (detailed in Section 5.1). All rendering calculations must strictly baseline against these virtual dimensions.*
- `TINCLE_FONT_S` : 16 (For small-sized fonts)
- `TINCLE_FONT_M` : 32 (For medium-sized fonts)
- `TINCLE_FONT_L` : 64 (For large-sized fonts)
- `TINCLE_MAX_STATES` : 256 (Maximum number of manageable states)

4.2. Mouse Button Definitions

- `TINCLE_MOUSE_LEFT` : 1 (Left button)
- `TINCLE_MOUSE_MIDDLE` : 2 (Middle wheel button)
- `TINCLE_MOUSE_RIGHT` : 3 (Right button)

4.3. Gamepad Definitions (Xbox Layout Standard)

- `TINGLE_PAD_A` : 0
- `TINGLE_PAD_B` : 1
- `TINGLE_PAD_X` : 2
- `TINGLE_PAD_Y` : 3
- `TINGLE_PAD_START` : 4
- `TINGLE_PAD_BACK` : 5
- `TINGLE_PAD_UP` : 6
- `TINGLE_PAD_DOWN` : 7
- `TINGLE_PAD_LEFT` : 8
- `TINGLE_PAD_RIGHT` : 9

4.4. Keyboard Scan Codes (SDL_scancode Compliant)

This section details the comprehensive list of hardware-independent scan codes utilized within TinCLE for keyboard input detection. These identifiers strictly align with standard standard `SDL_scancode` mapping values.

Alphabet Keys

Constant Name	Scan Code
<code>TINGLE_KEY_A</code>	4
<code>TINGLE_KEY_B</code>	5
<code>TINGLE_KEY_C</code>	6
<code>TINGLE_KEY_D</code>	7
<code>TINGLE_KEY_E</code>	8
<code>TINGLE_KEY_F</code>	9
<code>TINGLE_KEY_G</code>	10
<code>TINGLE_KEY_H</code>	11
<code>TINGLE_KEY_I</code>	12
<code>TINGLE_KEY_J</code>	13
<code>TINGLE_KEY_K</code>	14
<code>TINGLE_KEY_L</code>	15
<code>TINGLE_KEY_M</code>	16
<code>TINGLE_KEY_N</code>	17
<code>TINGLE_KEY_O</code>	18
<code>TINGLE_KEY_P</code>	19
<code>TINGLE_KEY_Q</code>	20
<code>TINGLE_KEY_R</code>	21

Constant Name	Scan Code
TINCLE_KEY_S	22
TINCLE_KEY_T	23
TINCLE_KEY_U	24
TINCLE_KEY_V	25
TINCLE_KEY_W	26
TINCLE_KEY_X	27
TINCLE_KEY_Y	28
TINCLE_KEY_Z	29

Numeric Keys (Top Row)

Constant Name	Scan Code
TINCLE_KEY_1	30
TINCLE_KEY_2	31
TINCLE_KEY_3	32
TINCLE_KEY_4	33
TINCLE_KEY_5	34
TINCLE_KEY_6	35
TINCLE_KEY_7	36
TINCLE_KEY_8	37
TINCLE_KEY_9	38
TINCLE_KEY_0	39

Core Control & Space Keys

Constant Name	Scan Code	Description
TINCLE_KEY_RETURN	40	Enter / Return
TINCLE_KEY_ESCAPE	41	Esc
TINCLE_KEY_BACKSPACE	42	Backspace
TINCLE_KEY_TAB	43	Tab
TINCLE_KEY_SPACE	44	Spacebar
TINCLE_KEY_CAPSLOCK	57	Caps Lock

Symbol Keys

Constant Name	Scan Code
TINCLE_KEY_MINUS	45
TINCLE_KEY_EQUALS	46
TINCLE_KEY_LEFTBRACKET	47
TINCLE_KEY_RIGHTBRACKET	48
TINCLE_KEY_BACKSLASH	49
TINCLE_KEY_NONUSHASH	50
TINCLE_KEY_SEMICOLON	51
TINCLE_KEY_APOSTROPHE	52
TINCLE_KEY_GRAVE	53
TINCLE_KEY_COMMA	54
TINCLE_KEY_PERIOD	55
TINCLE_KEY_SLASH	56

Navigation & Editing Keys

Constant Name	Scan Code
TINCLE_KEY_PRINTSCREEN	70
TINCLE_KEY_SCROLLLOCK	71
TINCLE_KEY_PAUSE	72
TINCLE_KEY_INSERT	73
TINCLE_KEY_HOME	74
TINCLE_KEY_PAGEUP	75
TINCLE_KEY_DELETE	76
TINCLE_KEY_END	77
TINCLE_KEY_PAGEDOWN	78

Arrow Keys

Constant Name	Scan Code
TINCLE_KEY_RIGHT	79
TINCLE_KEY_LEFT	80

Constant Name	Scan Code
TINCLE_KEY_DOWN	81
TINCLE_KEY_UP	82

Function Keys (F1 - F24)

Constant Name	Scan Code	Constant Name	Scan Code
TINCLE_KEY_F1	58	TINCLE_KEY_F13	104
TINCLE_KEY_F2	59	TINCLE_KEY_F14	105
TINCLE_KEY_F3	60	TINCLE_KEY_F15	106
TINCLE_KEY_F4	61	TINCLE_KEY_F16	107
TINCLE_KEY_F5	62	TINCLE_KEY_F17	108
TINCLE_KEY_F6	63	TINCLE_KEY_F18	109
TINCLE_KEY_F7	64	TINCLE_KEY_F19	110
TINCLE_KEY_F8	65	TINCLE_KEY_F20	111
TINCLE_KEY_F9	66	TINCLE_KEY_F21	112
TINCLE_KEY_F10	67	TINCLE_KEY_F22	113
TINCLE_KEY_F11	68	TINCLE_KEY_F23	114
TINCLE_KEY_F12	69	TINCLE_KEY_F24	115

Numeric Pad (Keypad) Keys

Constant Name	Scan Code	Description / Character
TINCLE_KEY_NUMLOCKCLEAR	83	Num Lock / Clear
TINCLE_KEY_KP_DIVIDE	84	Keypad /
TINCLE_KEY_KP_MULTIPLY	85	Keypad *
TINCLE_KEY_KP_MINUS	86	Keypad -
TINCLE_KEY_KP_PLUS	87	Keypad +
TINCLE_KEY_KP_ENTER	88	Keypad Enter
TINCLE_KEY_KP_1	89	Keypad 1
TINCLE_KEY_KP_2	90	Keypad 2
TINCLE_KEY_KP_3	91	Keypad 3
TINCLE_KEY_KP_4	92	Keypad 4

Constant Name	Scan Code	Description / Character
TINCLE_KEY_KP_5	93	Keypad 5
TINCLE_KEY_KP_6	94	Keypad 6
TINCLE_KEY_KP_7	95	Keypad 7
TINCLE_KEY_KP_8	96	Keypad 8
TINCLE_KEY_KP_9	97	Keypad 9
TINCLE_KEY_KP_0	98	Keypad 0
TINCLE_KEY_KP_PERIOD	99	Keypad .
TINCLE_KEY_KP_EQUALS	103	Keypad =
TINCLE_KEY_KP_COMMA	133	Keypad ,
TINCLE_KEY_KP_EQUALSAS400	134	AS400 Keypad =
TINCLE_KEY_KP_00	176	Keypad 00
TINCLE_KEY_KP_000	177	Keypad 000
TINCLE_KEY_THOUSANDSSEPARATOR	178	Keypad Thousands Separator
TINCLE_KEY_DECIMALSEPARATOR	179	Keypad Decimal Separator
TINCLE_KEY_CURRENCYUNIT	180	Keypad Currency Unit
TINCLE_KEY_CURRENCYSUBUNIT	181	Keypad Currency Sub-unit
TINCLE_KEY_KP_LEFTPAREN	182	Keypad (
TINCLE_KEY_KP_RIGHTPAREN	183	Keypad)
TINCLE_KEY_KP_LEFTBRACE	184	Keypad {
TINCLE_KEY_KP_RIGHTBRACE	185	Keypad }
TINCLE_KEY_KP_TAB	186	Keypad Tab
TINCLE_KEY_KP_BACKSPACE	187	Keypad Backspace
TINCLE_KEY_KP_A	188	Keypad Hex A
TINCLE_KEY_KP_B	189	Keypad Hex B
TINCLE_KEY_KP_C	190	Keypad Hex C
TINCLE_KEY_KP_D	191	Keypad Hex D
TINCLE_KEY_KP_E	192	Keypad Hex E
TINCLE_KEY_KP_F	193	Keypad Hex F
TINCLE_KEY_KP_XOR	194	Keypad XOR
TINCLE_KEY_KP_POWER	195	Keypad ^ (Power)

Constant Name	Scan Code	Description / Character
TINCLE_KEY_KP_PERCENT	196	Keypad %
TINCLE_KEY_KP_LESS	197	Keypad <
TINCLE_KEY_KP_GREATER	198	Keypad >
TINCLE_KEY_KP_AMPERSAND	199	Keypad &
TINCLE_KEY_KP_DBLAMPERSAND	200	Keypad &&
TINCLE_KEY_KP_VERTICALBAR	201	Keypad
TINCLE_KEY_KP_DBLVERTICALBAR	202	Keypad
TINCLE_KEY_KP_COLON	203	Keypad :
TINCLE_KEY_KP_HASH	204	Keypad #
TINCLE_KEY_KP_SPACE	205	Keypad Space
TINCLE_KEY_KP_AT	206	Keypad @
TINCLE_KEY_KP_EXCLAM	207	Keypad !
TINCLE_KEY_KP_MEMSTORE	208	Keypad Memory Store
TINCLE_KEY_KP_MEMRECALL	209	Keypad Memory Recall
TINCLE_KEY_KP_MEMCLEAR	210	Keypad Memory Clear
TINCLE_KEY_KP_MEMADD	211	Keypad Memory Add
TINCLE_KEY_KP_MEMSUBTRACT	212	Keypad Memory Subtract
TINCLE_KEY_KP_MEMMULTIPLY	213	Keypad Memory Multiply
TINCLE_KEY_KP_MEMDIVIDE	214	Keypad Memory Divide
TINCLE_KEY_KP_PLUSMINUS	215	Keypad +/-
TINCLE_KEY_KP_CLEAR	216	Keypad Clear
TINCLE_KEY_KP_CLEARENTRY	217	Keypad Clear Entry
TINCLE_KEY_KP_BINARY	218	Keypad Binary Mode
TINCLE_KEY_KP_OCTAL	219	Keypad Octal Mode
TINCLE_KEY_KP_DECIMAL	220	Keypad Decimal Mode
TINCLE_KEY_KP_HEXADECIMAL	221	Keypad Hexadecimal Mode

Modifier Keys

Constant Name	Scan Code	Description
TINCLE_KEY_LCTRL	224	Left Control

Constant Name	Scan Code	Description
TINCLE_KEY_LSHIFT	225	Left Shift
TINCLE_KEY_LALT	226	Left Alt / Option
TINCLE_KEY_LGUI	227	Left GUI (Windows / Command)
TINCLE_KEY_RCTRL	228	Right Control
TINCLE_KEY_RSHIFT	229	Right Shift
TINCLE_KEY_RALT	230	Right Alt / Option
TINCLE_KEY_RGUI	231	Right GUI (Windows / Command)

International & Language Keys

Constant Name	Scan Code	Constant Name	Scan Code
TINCLE_KEY_INTERNATIONAL1	135	TINCLE_KEY_LANG1	144
TINCLE_KEY_INTERNATIONAL2	136	TINCLE_KEY_LANG2	145
TINCLE_KEY_INTERNATIONAL3	137	TINCLE_KEY_LANG3	146
TINCLE_KEY_INTERNATIONAL4	138	TINCLE_KEY_LANG4	147
TINCLE_KEY_INTERNATIONAL5	139	TINCLE_KEY_LANG5	148
TINCLE_KEY_INTERNATIONAL6	140	TINCLE_KEY_LANG6	149
TINCLE_KEY_INTERNATIONAL7	141	TINCLE_KEY_LANG7	150
TINCLE_KEY_INTERNATIONAL8	142	TINCLE_KEY_LANG8	151
TINCLE_KEY_INTERNATIONAL9	143	TINCLE_KEY_LANG9	152

System, Media & Application Keys

Constant Name	Scan Code	Description / Use Case
TINCLE_KEY_UNKNOWN	0	Unrecognized / Edge cases
TINCLE_KEY_NONUSBACKSLASH	100	Non-US Backslash
TINCLE_KEY_APPLICATION	101	Context Menu Key
TINCLE_KEY_POWER	102	System Power Status
TINCLE_KEY_EXECUTE	116	Execute Command
TINCLE_KEY_HELP	117	Help Menu Trigger
TINCLE_KEY_MENU	118	Menu Key
TINCLE_KEY_SELECT	119	Select Command

Constant Name	Scan Code	Description / Use Case
TINCLE_KEY_STOP	120	Stop Operation
TINCLE_KEY_AGAIN	121	Redo / Repeat
TINCLE_KEY_UNDO	122	Undo Action
TINCLE_KEY_CUT	123	Clipboard Cut
TINCLE_KEY_COPY	124	Clipboard Copy
TINCLE_KEY_PASTE	125	Clipboard Paste
TINCLE_KEY_FIND	126	Search / Find
TINCLE_KEY_MUTE	127	Master Audio Mute (Legacy)
TINCLE_KEY_VOLUMEUP	128	System Volume Up
TINCLE_KEY_VOLUMEDOWN	129	System Volume Down
TINCLE_KEY_ALTERASE	153	Alt Erase
TINCLE_KEY_SYSREQ	154	Sys Req / Print Screen
TINCLE_KEY_CANCEL	155	Cancel Action
TINCLE_KEY_CLEAR	156	Clear Display / State
TINCLE_KEY_PRIOR	157	Prior Page / Selection
TINCLE_KEY_RETURN2	158	Alternative Return
TINCLE_KEY_SEPARATOR	159	Separator
TINCLE_KEY_OUT	160	Out
TINCLE_KEY_OPER	161	Operational Mode
TINCLE_KEY_CLEARAGAIN	162	Clear / Reset Again
TINCLE_KEY_CRSEL	163	CR Sel
TINCLE_KEY_EXSEL	164	EX Sel
TINCLE_KEY_MODE	257	Mode Switch
TINCLE_KEY_AUDIONEXT	258	Media Track Next
TINCLE_KEY_AUDIOPREV	259	Media Track Previous
TINCLE_KEY_AUDIOSTOP	260	Media Audio Stop
TINCLE_KEY_AUDIOPLAY	261	Media Audio Play / Pause
TINCLE_KEY_AUDIOMUTE	262	System Audio Mute
TINCLE_KEY_MEDIASELECT	263	Launch Media Player
TINCLE_KEY_WWW	264	Launch Web Browser

Constant Name	Scan Code	Description / Use Case
TINCLE_KEY_MAIL	265	Launch Mail Application
TINCLE_KEY_CALCULATOR	266	Launch Calculator
TINCLE_KEY_COMPUTER	267	Open My Computer / Files
TINCLE_KEY_AC_SEARCH	268	AC Search
TINCLE_KEY_AC_HOME	269	AC Home
TINCLE_KEY_AC_BACK	270	AC Back
TINCLE_KEY_AC_FORWARD	271	AC Forward
TINCLE_KEY_AC_STOP	272	AC Stop
TINCLE_KEY_AC_REFRESH	273	AC Refresh
TINCLE_KEY_AC_BOOKMARKS	274	AC Bookmarks
TINCLE_KEY_BRIGHTNESSDOWN	275	Monitor Brightness Down
TINCLE_KEY_BRIGHTNESSUP	276	Monitor Brightness Up
TINCLE_KEY_DISPLAYSWITCH	277	Display Output Toggle
TINCLE_KEY_KBDILLUMTOGGLE	278	Keyboard Backlight Toggle
TINCLE_KEY_KBDILLUMDOWN	279	Keyboard Backlight Down
TINCLE_KEY_KBDILLUMUP	280	Keyboard Backlight Up
TINCLE_KEY_EJECT	281	Optical Drive Eject
TINCLE_KEY_SLEEP	282	System Sleep Trigger
TINCLE_KEY_APP1	283	Launch Custom App 1
TINCLE_KEY_APP2	284	Launch Custom App 2
TINCLE_KEY_AUDIOREWIND	285	Media Audio Rewind
TINCLE_KEY_AUDIOFASTFORWARD	286	Media Audio Fast Forward
TINCLE_KEY_SOFTLEFT	287	Soft Left Control
TINCLE_KEY_SOFTRIGHT	288	Soft Right Control
TINCLE_KEY_CALL	289	Initiate Call / Accept
TINCLE_KEY_ENDCALL	290	Terminate Call / Decline

4.5. Color Structure & Preset Definitions

This section defines a custom color structure that stores RGBA channels in 1-byte integers, alongside handy preset macros for rapid color selection.

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

【Quick-Use Macros (Compound Literals)】

These convenient preset macros can be injected inline directly into function arguments.

1. PERFECT MONOTONE SYSTEM (19 Steps Gray Scale)

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_BLACK	(TinCLE_Color){0, 0, 0, 255}	TINCLE_COLOR_GRAY_1	(TinCLE_Color){13, 13, 13, 255}
TINCLE_COLOR_GRAY_2	(TinCLE_Color){26, 26, 26, 255}	TINCLE_COLOR_GRAY_3	(TinCLE_Color){39, 39, 39, 255}
TINCLE_COLOR_GRAY_4	(TinCLE_Color){51, 51, 51, 255}	TINCLE_COLOR_GRAY_5	(TinCLE_Color){64, 64, 64, 255}
TINCLE_COLOR_GRAY_6	(TinCLE_Color){77, 77, 77, 255}	TINCLE_COLOR_GRAY_7	(TinCLE_Color){90, 90, 90, 255}
TINCLE_COLOR_GRAY_8	(TinCLE_Color){102, 102, 102, 255}	TINCLE_COLOR_GRAY_9	(TinCLE_Color){115, 115, 115, 255}
TINCLE_COLOR_GRAY	(TinCLE_Color){128, 128, 128, 255}	TINCLE_COLOR_GRAY_10	(TinCLE_Color){140, 140, 140, 255}
TINCLE_COLOR_GRAY_11	(TinCLE_Color){153, 153, 153, 255}	TINCLE_COLOR_GRAY_12	(TinCLE_Color){166, 166, 166, 255}
TINCLE_COLOR_GRAY_13	(TinCLE_Color){179, 179, 179, 255}	TINCLE_COLOR_GRAY_14	(TinCLE_Color){204, 204, 204, 255}
TINCLE_COLOR_GRAY_15	(TinCLE_Color){217, 217, 217, 255}	TINCLE_COLOR_GRAY_16	(TinCLE_Color){230, 230, 230, 255}
TINCLE_COLOR_GRAY_17	(TinCLE_Color){243, 243, 243, 255}	TINCLE_COLOR_WHITE	(TinCLE_Color){255, 255, 255, 255}

2. REGULAR RETRO 16 COLORS

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_NAVY	(TinCLE_Color){0, 0, 128, 255}	TINCLE_COLOR_DARK_BLUE	(TinCLE_Color){0, 0, 139, 255}
TINCLE_COLOR_GREEN	(TinCLE_Color){0, 128, 0, 255}	TINCLE_COLOR_TEAL	(TinCLE_Color){0, 128, 128, 255}
TINCLE_COLOR_MAROON	(TinCLE_Color){128, 0, 0, 255}	TINCLE_COLOR_PURPLE	(TinCLE_Color){128, 0, 128, 255}

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_OLIVE	(TincLE_Color){128, 128, 0, 255}	TINCLE_COLOR_LIGHT_GRAY	(TincLE_Color){211, 211, 211, 255}
TINCLE_COLOR_DARK_GRAY	(TincLE_Color){169, 169, 169, 255}	TINCLE_COLOR_BLUE	(TincLE_Color){0, 0, 255, 255}
TINCLE_COLOR_LIME	(TincLE_Color){0, 255, 0, 255}	TINCLE_COLOR_CYAN	(TincLE_Color){0, 255, 255, 255}
TINCLE_COLOR_RED	(TincLE_Color){255, 0, 0, 255}	TINCLE_COLOR_MAGENTA	(TincLE_Color){255, 0, 255, 255}
TINCLE_COLOR_YELLOW	(TincLE_Color){255, 255, 0, 255}		

3. RED / ORANGE / BROWN SYSTEMS (35 Colors)

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_DARK_RED	(TincLE_Color){139, 0, 0, 255}	TINCLE_COLOR_BROWN	(TincLE_Color){165, 42, 42, 255}
TINCLE_COLOR_FIREBRICK	(TincLE_Color){178, 34, 34, 255}	TINCLE_COLOR_CRIMSON	(TincLE_Color){220, 20, 60, 255}
TINCLE_COLOR_TOMATO	(TincLE_Color){255, 99, 71, 255}	TINCLE_COLOR_CORAL	(TincLE_Color){255, 127, 80, 255}
TINCLE_COLOR_INDIAN_RED	(TincLE_Color){205, 92, 92, 255}	TINCLE_COLOR_LIGHT_CORAL	(TincLE_Color){240, 128, 128, 255}
TINCLE_COLOR_DARK_SALMON	(TincLE_Color){233, 150, 122, 255}	TINCLE_COLOR_SALMON	(TincLE_Color){250, 128, 114, 255}
TINCLE_COLOR_LIGHT_SALMON	(TincLE_Color){255, 160, 122, 255}	TINCLE_COLOR_ORANGE_RED	(TincLE_Color){255, 69, 0, 255}
TINCLE_COLOR_DARK_ORANGE	(TincLE_Color){255, 140, 0, 255}	TINCLE_COLOR_ORANGE	(TincLE_Color){255, 165, 0, 255}
TINCLE_COLOR_GOLD	(TincLE_Color){255, 215, 0, 255}	TINCLE_COLOR_SADDLE_BROWN	(TincLE_Color){139, 69, 19, 255}
TINCLE_COLOR_SIENNA	(TincLE_Color){160, 82, 45, 255}	TINCLE_COLOR_CHOCOLATE	(TincLE_Color){210, 105, 30, 255}
TINCLE_COLOR_PERU	(TincLE_Color){205, 133, 63, 255}	TINCLE_COLOR_SANDY_BROWN	(TincLE_Color){244, 164, 96, 255}
TINCLE_COLOR_BURLEY_WOOD	(TincLE_Color){222, 184, 135, 255}	TINCLE_COLOR_TAN	(TincLE_Color){210, 180, 140, 255}
TINCLE_COLOR_ROSY_BROWN	(TincLE_Color){188, 143, 143, 255}	TINCLE_COLOR_MOCCASIN	(TincLE_Color){255, 228, 181, 255}

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_PEACH_PUFF	(TinCLE_Color) {255, 218, 185, 255}	TINCLE_COLOR_NAVAJO_WHITE	(TinCLE_Color) {255, 222, 173, 255}
TINCLE_COLOR_BISQUE	(TinCLE_Color) {255, 228, 196, 255}	TINCLE_COLOR_BRICK_RED	(TinCLE_Color) {153, 51, 51, 255}
TINCLE_COLOR_DEEP_ORANGE	(TinCLE_Color) {255, 102, 0, 255}	TINCLE_COLOR_PUMPKIN	(TinCLE_Color) {204, 102, 0, 255}
TINCLE_COLOR_RUST	(TinCLE_Color) {153, 102, 51, 255}	TINCLE_COLOR_COPPER	(TinCLE_Color) {204, 102, 51, 255}
TINCLE_COLOR_AMBER	(TinCLE_Color) {255, 153, 0, 255}	TINCLE_COLOR_OCHRE	(TinCLE_Color) {204, 153, 51, 255}
TINCLE_COLOR_MAHOGANY	(TinCLE_Color) {102, 0, 0, 255}		

4. YELLOW / GREEN / EARTH SYSTEMS (45 Colors)

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_DARK_KHAKI	(TinCLE_Color) {189, 183, 107, 255}	TINCLE_COLOR_KHAKI	(TinCLE_Color) {240, 230, 140, 255}
TINCLE_COLOR_PALE_GOLDENROD	(TinCLE_Color) {238, 232, 170, 255}	TINCLE_COLOR_LIGHT_GOLDENROD_YELLOW	(TinCLE_Color) {250, 250, 210, 255}
TINCLE_COLOR_LIGHT_YELLOW	(TinCLE_Color) {255, 255, 224, 255}	TINCLE_COLOR_LEMON_CHIFFON	(TinCLE_Color) {255, 250, 205, 255}
TINCLE_COLOR_PAPAYA_WHIP	(TinCLE_Color) {255, 239, 213, 255}	TINCLE_COLOR_CORNSILK	(TinCLE_Color) {255, 248, 220, 255}
TINCLE_COLOR_DARK_GREEN	(TinCLE_Color) {0, 100, 0, 255}	TINCLE_COLOR_FOREST_GREEN	(TinCLE_Color) {34, 139, 34, 255}
TINCLE_COLOR_SEA_GREEN	(TinCLE_Color) {46, 139, 87, 255}	TINCLE_COLOR_MEDIUM_SEA_GREEN	(TinCLE_Color) {60, 179, 113, 255}
TINCLE_COLOR_PALE_GREEN	(TinCLE_Color) {152, 251, 152, 255}	TINCLE_COLOR_LIGHT_GREEN	(TinCLE_Color) {144, 238, 144, 255}
TINCLE_COLOR_SPRING_GREEN	(TinCLE_Color) {0, 255, 127, 255}	TINCLE_COLOR_MEDIUM_SPRING_GREEN	(TinCLE_Color) {0, 250, 154, 255}
TINCLE_COLOR_LAWN_GREEN	(TinCLE_Color) {124, 252, 0, 255}	TINCLE_COLOR_CHARTREUSE	(TinCLE_Color) {127, 255, 0, 255}
TINCLE_COLOR_GREEN_YELLOW	(TinCLE_Color) {173, 255, 47, 255}	TINCLE_COLOR_LIME_GREEN	(TinCLE_Color) {50, 205, 50, 255}
TINCLE_COLOR_YELLOW_GREEN	(TinCLE_Color) {154, 205, 50, 255}	TINCLE_COLOR_OLIVE_DRAB	(TinCLE_Color) {107, 142, 35, 255}

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_DARK_OLIVE_GREEN	(TinCLE_Color) {85, 107, 47, 255}	TINCLE_COLOR_MINT_GREEN	(TinCLE_Color) {102, 255, 153, 255}
TINCLE_COLOR_MOSS_GREEN	(TinCLE_Color) {102, 153, 102, 255}	TINCLE_COLOR_SAGE	(TinCLE_Color) {153, 204, 153, 255}
TINCLE_COLOR_EMERALD	(TinCLE_Color) {0, 204, 102, 255}	TINCLE_COLOR_PINE_GREEN	(TinCLE_Color) {0, 102, 51, 255}
TINCLE_COLOR_JADE	(TinCLE_Color) {0, 153, 102, 255}	TINCLE_COLOR_BRIGHT_GREEN	(TinCLE_Color) {102, 255, 0, 255}
TINCLE_COLOR_PASTEL_GREEN	(TinCLE_Color) {204, 255, 204, 255}	TINCLE_COLOR_ARMY_GREEN	(TinCLE_Color) {51, 102, 51, 255}
TINCLE_COLOR_SEA_FOAM	(TinCLE_Color) {153, 255, 204, 255}	TINCLE_COLOR_PALE_YELLOW	(TinCLE_Color) {255, 255, 153, 255}
TINCLE_COLOR_BANANA	(TinCLE_Color) {255, 255, 102, 255}	TINCLE_COLOR_MUSTARD	(TinCLE_Color) {204, 204, 51, 255}
TINCLE_COLOR_OLIVE_GREEN	(TinCLE_Color) {102, 102, 0, 255}	TINCLE_COLOR_BRASS	(TinCLE_Color) {153, 153, 51, 255}
TINCLE_COLOR_DESERT_SAND	(TinCLE_Color) {204, 204, 153, 255}	TINCLE_COLOR_ECRU	(TinCLE_Color) {204, 153, 102, 255}
TINCLE_COLOR_CAMEL	(TinCLE_Color) {153, 102, 0, 255}	TINCLE_COLOR_SEPIA	(TinCLE_Color) {102, 51, 0, 255}
TINCLE_COLOR_CHOCOLATE_DARK	(TinCLE_Color) {51, 0, 0, 255}	TINCLE_COLOR_DEEP_FOREST	(TinCLE_Color) {0, 51, 0, 255}
TINCLE_COLOR_NIGHT_GREEN	(TinCLE_Color) {0, 51, 51, 255}		

5. CYAN / AQUA / BLUE SYSTEMS (50 Colors)

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_DARK_CYAN	(TinCLE_Color) {0, 139, 139, 255}	TINCLE_COLOR_LIGHT_SEA_GREEN	(TinCLE_Color) {32, 178, 170, 255}
TINCLE_COLOR_CADET_BLUE	(TinCLE_Color) {95, 158, 160, 255}	TINCLE_COLOR_DARK_TURQUOISE	(TinCLE_Color) {0, 206, 209, 255}
TINCLE_COLOR_MEDIUM_TURQUOISE	(TinCLE_Color) {72, 209, 204, 255}	TINCLE_COLOR_TURQUOISE	(TinCLE_Color) {64, 224, 208, 255}
TINCLE_COLOR_PALE_TURQUOISE	(TinCLE_Color) {175, 238, 238, 255}	TINCLE_COLOR_LIGHT_CYAN	(TinCLE_Color) {224, 255, 255, 255}
TINCLE_COLOR_AZURE	(TinCLE_Color) {240, 255, 255, 255}	TINCLE_COLOR_POWDER_BLUE	(TinCLE_Color) {176, 224, 230, 255}

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_LIGHT_BLUE	(TincLE_Color) {173, 216, 230, 255}	TINCLE_COLOR_SKY_BLUE	(TincLE_Color) {135, 206, 235, 255}
TINCLE_COLOR_LIGHT_SKY_BLUE	(TincLE_Color) {135, 206, 250, 255}	TINCLE_COLOR_DEEP_SKY_BLUE	(TincLE_Color) {0, 191, 255, 255}
TINCLE_COLOR_DODGER_BLUE	(TincLE_Color) {30, 144, 255, 255}	TINCLE_COLOR_CORNFLOWER_BLUE	(TincLE_Color) {100, 149, 237, 255}
TINCLE_COLOR_STEEL_BLUE	(TincLE_Color) {70, 130, 180, 255}	TINCLE_COLOR_ROYAL_BLUE	(TincLE_Color) {65, 105, 225, 255}
TINCLE_COLOR_MEDIUM_BLUE	(TincLE_Color) {0, 0, 205, 255}	TINCLE_COLOR_MIDNIGHT_BLUE	(TincLE_Color) {25, 25, 112, 255}
TINCLE_COLOR_ICE_BLUE	(TincLE_Color) {204, 255, 255, 255}	TINCLE_COLOR_BABY_BLUE	(TincLE_Color) {153, 204, 255, 255}
TINCLE_COLOR_ELECTRIC_BLUE	(TincLE_Color) {0, 204, 255, 255}	TINCLE_COLOR_DEEP_BLUE	(TincLE_Color) {0, 102, 255, 255}
TINCLE_COLOR_COBALT	(TincLE_Color) {0, 51, 204, 255}	TINCLE_COLOR_OCEAN_BLUE	(TincLE_Color) {0, 102, 153, 255}
TINCLE_COLOR_MARINE_BLUE	(TincLE_Color) {0, 51, 153, 255}	TINCLE_COLOR_PASTEL_BLUE	(TincLE_Color) {153, 153, 255, 255}
TINCLE_COLOR_DENIM	(TincLE_Color) {51, 102, 153, 255}	TINCLE_COLOR_SLATE_LIGHT	(TincLE_Color) {102, 153, 204, 255}
TINCLE_COLOR_AQUA_GREEN	(TincLE_Color) {51, 204, 153, 255}	TINCLE_COLOR_DARK_AQUA	(TincLE_Color) {0, 102, 102, 255}
TINCLE_COLOR_CYAN_DEEP	(TincLE_Color) {0, 153, 153, 255}	TINCLE_COLOR_TEAL_LIGHT	(TincLE_Color) {51, 153, 153, 255}
TINCLE_COLOR_NIGHT_BLUE	(TincLE_Color) {0, 0, 51, 255}	TINCLE_COLOR_SAPPHIRE	(TincLE_Color) {15, 82, 186, 255}
TINCLE_COLOR_ULTRAMARINE	(TincLE_Color) {18, 10, 143, 255}	TINCLE_COLOR_LAPIS	(TincLE_Color) {38, 97, 156, 255}
TINCLE_COLOR_CERULEAN	(TincLE_Color) {0, 123, 167, 255}	TINCLE_COLOR_INDIGO_DEEP	(TincLE_Color) {0, 46, 93, 255}
TINCLE_COLOR_CYAN_PALE	(TincLE_Color) {153, 204, 204, 255}	TINCLE_COLOR_BLUE_PALE	(TincLE_Color) {204, 204, 255, 255}
TINCLE_COLOR_AQUA_PALE	(TincLE_Color) {204, 255, 204, 255}	TINCLE_COLOR_SKY_PALE	(TincLE_Color) {153, 255, 255, 255}
TINCLE_COLOR_ROBINS_EGG	(TincLE_Color) {0, 204, 153, 255}	TINCLE_COLOR_MIDNIGHT_DEEP	(TincLE_Color) {10, 10, 40, 255}

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_VOID_BLUE	(TincLE_Color) {5, 5, 25, 255}	TINCLE_COLOR_PACIFIC_BLUE	(TincLE_Color) {26, 153, 204, 255}
TINCLE_COLOR_STORM_BLUE	(TincLE_Color) {102, 102, 153, 255}	TINCLE_COLOR_SHADOW_BLUE	(TincLE_Color) {51, 51, 102, 255}

6. PURPLE / PINK / LUXURY COSMETICS (50 Colors)

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_INDIGO	(TincLE_Color) {75, 0, 130, 255}	TINCLE_COLOR_DARK_SLATE_BLUE	(TincLE_Color) {72, 61, 139, 255}
TINCLE_COLOR_SLATE_BLUE	(TincLE_Color) {106, 90, 205, 255}	TINCLE_COLOR_MEDIUM_SLATE_BLUE	(TincLE_Color) {123, 104, 238, 255}
TINCLE_COLOR_MEDIUM_PURPLE	(TincLE_Color) {147, 112, 219, 255}	TINCLE_COLOR_DARK_MAGENTA	(TincLE_Color) {139, 0, 139, 255}
TINCLE_COLOR_DARK_VIOLET	(TincLE_Color) {148, 0, 211, 255}	TINCLE_COLOR_DARK_ORCHID	(TincLE_Color) {153, 50, 204, 255}
TINCLE_COLOR_MEDIUM_ORCHID	(TincLE_Color) {186, 85, 211, 255}	TINCLE_COLOR_ORCHID	(TincLE_Color) {218, 112, 214, 255}
TINCLE_COLOR_VIOLET	(TincLE_Color) {238, 130, 238, 255}	TINCLE_COLOR_PLUM	(TincLE_Color) {221, 160, 221, 255}
TINCLE_COLOR_THISTLE	(TincLE_Color) {216, 191, 216, 255}	TINCLE_COLOR_LAVENDER	(TincLE_Color) {230, 230, 250, 255}
TINCLE_COLOR_DEEP_PINK	(TincLE_Color) {255, 20, 147, 255}	TINCLE_COLOR_HOT_PINK	(TincLE_Color) {255, 105, 180, 255}
TINCLE_COLOR_PALE_VIOLET_RED	(TincLE_Color) {219, 112, 147, 255}	TINCLE_COLOR_MEDIUM_VIOLET_RED	(TincLE_Color) {199, 21, 133, 255}
TINCLE_COLOR_LIGHT_PINK	(TincLE_Color) {255, 182, 193, 255}	TINCLE_COLOR_PINK	(TincLE_Color) {255, 192, 203, 255}
TINCLE_COLOR_MISTY_ROSE	(TincLE_Color) {255, 228, 225, 255}	TINCLE_COLOR_LAVENDER_PALE	(TincLE_Color) {204, 153, 255, 255}
TINCLE_COLOR_VIOLET_BRIGHT	(TincLE_Color) {204, 0, 255, 255}	TINCLE_COLOR_MAGENTA_DARK	(TincLE_Color) {153, 0, 153, 255}
TINCLE_COLOR_PURPLE_DARK	(TincLE_Color) {102, 0, 102, 255}	TINCLE_COLOR_AMETHYST	(TincLE_Color) {153, 102, 204, 255}
TINCLE_COLOR_BOYSENBERRY	(TincLE_Color) {102, 0, 51, 255}	TINCLE_COLOR_MAUVE	(TincLE_Color) {204, 153, 204, 255}
TINCLE_COLOR_PASTEL_PINK	(TincLE_Color) {255, 204, 204, 255}	TINCLE_COLOR_PASTEL_PURPLE	(TincLE_Color) {204, 204, 255, 255}

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_WINE	(TincLE_Color) {102, 51, 51, 255}	TINCLE_COLOR_BURGUNDY	(TincLE_Color) {128, 0, 32, 255}
TINCLE_COLOR_LILAC	(TincLE_Color) {200, 162, 200, 255}	TINCLE_COLOR_REDNUM	(TincLE_Color) {51, 0, 0, 255}
TINCLE_COLOR_VOID_PURPLE	(TincLE_Color) {25, 5, 25, 255}	TINCLE_COLOR_FUSCHIA_PALE	(TincLE_Color) {255, 153, 255, 255}
TINCLE_COLOR_ROSE_DEEP	(TincLE_Color) {255, 51, 153, 255}	TINCLE_COLOR_ROSE_DARK	(TincLE_Color) {153, 51, 102, 255}
TINCLE_COLOR_ROSE_PALE	(TincLE_Color) {255, 204, 221, 255}	TINCLE_COLOR_BERRY	(TincLE_Color) {204, 0, 102, 255}
TINCLE_COLOR_MULBERRY	(TincLE_Color) {153, 51, 153, 255}	TINCLE_COLOR_RAISIN	(TincLE_Color) {51, 25, 51, 255}
TINCLE_COLOR_NIGHT_PURPLE	(TincLE_Color) {51, 0, 51, 255}	TINCLE_COLOR_BUBBLEGUM	(TincLE_Color) {255, 102, 204, 255}
TINCLE_COLOR_MAGENTA_PALE	(TincLE_Color) {255, 102, 255, 255}	TINCLE_COLOR_ORCHID_DARK	(TincLE_Color) {102, 51, 102, 255}
TINCLE_COLOR_PLUM_DARK	(TincLE_Color) {153, 102, 153, 255}	TINCLE_COLOR_SANGRIA	(TincLE_Color) {153, 0, 51, 255}
TINCLE_COLOR_VIOLET_PALE	(TincLE_Color) {238, 180, 238, 255}	TINCLE_COLOR_BLUSH	(TincLE_Color) {222, 93, 131, 255}

7. SPECIAL ARCHITECTURE & GHOST DETAILS (41 Colors)

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_SLATE_GRAY	(TincLE_Color) {112, 128, 144, 255}	TINCLE_COLOR_LIGHT_SLATE_GRAY	(TincLE_Color) {119, 136, 153, 255}
TINCLE_COLOR_DARK_SLATE_GRAY	(TincLE_Color) {47, 79, 79, 255}	TINCLE_COLOR_ALICE_BLUE	(TincLE_Color) {240, 248, 255, 255}
TINCLE_COLOR_GHOST_WHITE	(TincLE_Color) {248, 248, 255, 255}	TINCLE_COLOR_SEASHELL	(TincLE_Color) {255, 245, 238, 255}
TINCLE_COLOR_WHITE_SMOKE	(TincLE_Color) {245, 245, 245, 255}	TINCLE_COLOR_BEIGE	(TincLE_Color) {245, 245, 220, 255}
TINCLE_COLOR_OLD_LACE	(TincLE_Color) {253, 245, 230, 255}	TINCLE_COLOR_FLORAL_WHITE	(TincLE_Color) {255, 250, 240, 255}
TINCLE_COLOR_IVORY	(TincLE_Color) {255, 255, 240, 255}	TINCLE_COLOR_ANTIQUÉ_WHITE	(TincLE_Color) {250, 235, 215, 255}
TINCLE_COLOR_LINEN	(TincLE_Color) {250, 240, 230, 255}	TINCLE_COLOR_LAVENDER_BLUSH	(TincLE_Color) {255, 240, 245, 255}

Macro Name	Value	Macro Name	Value
TINCLE_COLOR_MINT_CREAM	(TincLE_Color) {245, 255, 250, 255}	TINCLE_COLOR_HONEYDEW	(TincLE_Color) {240, 255, 240, 255}
TINCLE_COLOR_SNOW	(TincLE_Color) {255, 250, 250, 255}	TINCLE_COLOR_SLATE_DARK	(TincLE_Color) {51, 51, 51, 255}
TINCLE_COLOR_SLATE_MEDIUM	(TincLE_Color) {102, 102, 102, 255}	TINCLE_COLOR_SLATE_PALE	(TincLE_Color) {153, 153, 153, 255}
TINCLE_COLOR_STEEL_DARK	(TincLE_Color) {51, 51, 102, 255}	TINCLE_COLOR_STEEL_PALE	(TincLE_Color) {153, 153, 204, 255}
TINCLE_COLOR_CHARCOAL	(TincLE_Color) {30, 30, 30, 255}	TINCLE_COLOR_GUNMETAL	(TincLE_Color) {42, 52, 57, 255}
TINCLE_COLOR ASPHALT	(TincLE_Color) {16, 16, 26, 255}	TINCLE_COLOR PEWTER	(TincLE_Color) {139, 141, 132, 255}
TINCLE_COLOR_PLATINUM	(TincLE_Color) {229, 228, 226, 255}	TINCLE_COLOR SILVER	(TincLE_Color) {192, 192, 192, 255}
TINCLE_COLOR_METALLIC_GRAY	(TincLE_Color) {102, 102, 153, 255}	TINCLE_COLOR COOL_GRAY	(TincLE_Color) {153, 153, 179, 255}
TINCLE_COLOR_WARM_GRAY	(TincLE_Color) {179, 179, 153, 255}	TINCLE_COLOR BONE	(TincLE_Color) {227, 218, 201, 255}
TINCLE_COLOR_PARCHMENT	(TincLE_Color) {245, 245, 220, 255}	TINCLE_COLOR PEARL	(TincLE_Color) {234, 230, 202, 255}
TINCLE_COLOR_ALABASTER	(TincLE_Color) {242, 240, 235, 255}	TINCLE_COLOR PALE_ROSE	(TincLE_Color) {255, 221, 221, 255}
TINCLE_COLOR_PALE_ORANGE	(TincLE_Color) {255, 221, 153, 255}	TINCLE_COLOR PALE_LIME	(TincLE_Color) {221, 255, 153, 255}
TINCLE_COLOR_PALE_AQUA	(TincLE_Color) {153, 255, 221, 255}	TINCLE_COLOR PALE_SKY	(TincLE_Color) {153, 221, 255, 255}
TINCLE_COLOR_PALE_LAVENDER	(TincLE_Color) {221, 153, 255, 255}		

8. GLOBAL ALIAS DEFINITIONS (For High-Speed Prototyping)

Alias Macro Name	Maps To	Alias Macro Name	Maps To
TINCLE_COLOR_BLK	TINCLE_COLOR_BLACK	TINCLE_COLOR_WHT	TINCLE_COLOR_WHITE
TINCLE_COLOR_GRN	TINCLE_COLOR_GREEN	TINCLE_COLOR_BLU	TINCLE_COLOR_BLUE
TINCLE_COLOR_YEL	TINCLE_COLOR_YELLOW	TINCLE_COLOR_MAG	TINCLE_COLOR_MAGENTA
TINCLE_COLOR_AQUA	TINCLE_COLOR_CYAN	TINCLE_COLOR_FUCHSIA	TINCLE_COLOR_MAGENTA

5. Full API Reference (Function Specifications)

This section provides the comprehensive API reference available when writing game logic within `game.c`. *Note: Although these functions are exported via `__declspec(dllexport)` within the engine (`tincle.dll`), they can be invoked seamlessly as standard C functions on the game application side.*

5.1. Lifecycle & System Control

```
void tincle_start(TinCLE_InitCallback init_cb, const char* title, bool is_debug, int width, int height);
```

- **Description:** The most critical function of the engine. It automatically executes environment initialization, asset loading, transitions into the main game loop, and handles terminal cleanup operations in a single one-shot sequence.
- **Arguments:**
 - `init_cb`: A user-defined callback function pointer tasked with initial loading and asset setups (typically, pass `game_load`).
 - `title`: A string literal specifying the window title text.
 - `is_debug`: A boolean toggle for Debug Mode. When set to `true`, the engine automatically spawns and docks a standard Windows command prompt window (Debug Console) to stream standard outputs such as `printf`. **Set to `false` for the production/release build to hide the command console completely.**
 - `width / height`: The virtual target resolution dimensions of the game canvas (strictly defaulted to `640` and `480`).

```
void tincle_add_state(int state_id, TinCLE_UpdateCallback update_cb, TinCLE_DrawCallback draw_cb);
```

- **Description:** Binds logic updating and rendering drawing callbacks to a specific game state identifier, executed sequentially every frame when the target state is active.
- **Arguments:**
 - `state_id`: An arbitrary integer constant or a user-defined enumeration value representing the state.
 - `update_cb`: A logic update function pointer adhering strictly to the signature: `void update(float dt);`.
 - `draw_cb`: A rendering function pointer adhering strictly to the signature: `void draw(void);`.

```
void tincle_change_state(int state_id);
```

- **Description:** Instantly switches the currently active game scene (active state) to the designated state. **CRITICAL CRITERIA:** You must call this function at the very end of your `game_load()` sequence to activate the initial game state, otherwise the viewport will remain indefinitely black without starting.
- **Arguments:**
 - `state_id`: A pre-registered state identifier to transition into.

5.2. Input (Device Control)

```
int tincle_key_pressed(int scancode);
```

- **Description:** Queries whether a specific keyboard key is currently being held down.
- **Arguments:**
 - `scancode`: A hardware-independent key identification code. **Crucial:** You must strictly pass the `TINCLE_KEY_*` constant macros defined in Section 4.4 (e.g., `TINCLE_KEY_SPACE`, `TINCLE_KEY_RIGHT`). Do not pass raw SDL internal scan codes.

```
void tincle_get_mouse(int* x, int* y);
```

- **Description:** Retrieves the current spatial coordinates of the mouse cursor, dynamically translated and scaled to align with the game's internal virtual resolution.
- **Arguments:**
- `x / y` : Integer pointers where the mapped virtual coordinates will be written.

```
int tincle_mouse_pressed(int button_type);
```

- **Description:** Queries whether a specific mouse button is currently being held down.
- **Arguments:**
- `button_type` : The preset mouse button identification macro defined in Section 4.2 (e.g., `TINCLE_MOUSE_LEFT`, `TINCLE_MOUSE_MIDDLE`, `TINCLE_MOUSE_RIGHT`).

```
int tincle_pad_pressed(int pad_button);
```

- **Description:** Queries whether a specific button on a connected gamepad is currently being held down. Input tracking is automatically normalized using standard Xbox controller layouts.
- **Arguments:**
- `pad_button` : The gamepad button macro defined in Section 4.3 (e.g., `TINCLE_PAD_A`, `TINCLE_PAD_START`, `TINCLE_PAD_LEFT`).

```
float tincle_pad_axis_x(void);
```

- **Description:** Measures the current horizontal (X-axis) tilt of the gamepad's left analog stick, returning a floating-point value clamped between `-1.0f` (full left) and `1.0f` (full right).
- **Notes for AI:** The engine internally implements a deadzone filter (clamping any absolute mechanical values below 5000 directly to `0.0f`). AI agents can safely map this return value directly into movement calculations without factoring in controller drift counter-measures.

```
float tincle_pad_axis_y(void);
```

- **Description:** Measures the current vertical (Y-axis) tilt of the gamepad's left analog stick, returning a floating-point value clamped between `-1.0f` (full up) and `1.0f` (full down).
- **Notes for AI:** Implements the exact same integrated deadzone optimization as the horizontal axis tracker.

5.3. Graphics & Rendering Control

```
void tincle_clear(TinCLE_Color color);
```

- **Description:** Fills and clears the entire screen with the specified color. This function **must without exception** be invoked at the absolute beginning of the `draw` callback function on every single frame.
- **Arguments:**
- `color` : A `TinCLE_Color` structural data instance (or one of the preset color macros defined in Section 4.5).

```
void tincle_load_font(const char* filename);
```

- **Description:** Loads a TrueType font file (`.ttf`) to establish the internal standard text rendering pipeline. TinCLE maintains a single, high-performance global font handle. Calling this function multiple times will automatically release the previously loaded font and overwrite it with the new asset.
- **Arguments:**
- `filename` : The relative path or filename of the `.ttf` asset (supports automatic extraction from packaged `game.data` distribution archives).

```
void tinkle_draw_text(const char* text, int x, int y, int size_type,
TinCLE_Color color);
```

- **Description:** Renders crisp, anti-aliased typographic text on the screen with native UTF-8 multi-byte character encoding support.
- **Arguments:**
 - `text` : The string literal or character array to be rendered.
 - `x / y` : The starting spatial coordinates on the internal virtual resolution canvas.
 - `size_type` : The designated font sizing token macro defined in Section 4.1 (`TINCLE_FONT_S` , `TINCLE_FONT_M` , or `TINCLE_FONT_L`).
 - `color` : The structural color properties used to tint the text characters.

```
void tinkle_draw_text_center(const char* text, int x, int y, int size_type,
TinCLE_Color color);
```

- **Description:** Renders crisp, anti-aliased typographic text horizontally and vertically centered at the specified spatial coordinates on the internal virtual resolution canvas. This is highly optimized for titles, UI labels, or button text where precise geometric centering is required. Fully supports native UTF-8 multi-byte character encoding.
- **Arguments:**
 - `text` : The string literal or character array to be rendered.
 - `x / y` : The target spatial coordinates that act as the exact central anchor point of the rendered text block.
 - `size_type` : The designated font sizing token macro defined in Section 4.1 (`TINCLE_FONT_S` , `TINCLE_FONT_M` , or `TINCLE_FONT_L`).
 - `color` : The structural color properties used to tint the text characters.

```
int tinkle_load_image(const char* filename);
```

- **Description:** Ingests a graphical image file (e.g., PNG, JPG) into the hardware rendering pipeline and caches it as an optimized texture resource.
- **Arguments:**
 - `filename` : The filename or directory path of the source image asset (fully optimized to auto-extract from packaged `game.data` files seamlessly).
- **Return Value:** Returns a uniquely assigned structural integer identifier (Texture ID) tracking the asset. Returns `-1` if the asset initialization or loading routine fails.

```
void tinkle_draw_image(int img_id, int x, int y);
```

- **Description:** Renders a pre-loaded texture resource onto the virtual canvas surface at its original unscaled (100% / 1:1) default dimensions.
- **Arguments:**
 - `img_id` : The cached texture index integer originally retrieved via `tinkle_load_image` .
 - `x / y` : The spatial alignment coordinates on the internal virtual resolution layout.

```
void tinkle_draw_image_ex(int img_id, int x, int y, int w, int h, double
angle);
```

- **Description:** Executes advanced, high-performance rendering configurations including spatial scaling (stretching/shrinking dimensions) and hardware rotational transformations.
- **Arguments:**
 - `img_id` : The target texture source index integer.
 - `x / y` : The spatial coordinate position tracking the upper-left (top-left) anchor point of the target rendering bounding box.

- `w / h` : The dynamic width and height scaling dimensions applied to the output render.
 - `angle` : The rotational angle calculated in degrees clockwise (0 to 360).
-

5.4. Audio (Music & Sound Effects) Control

```
void tincle_play_music(const char* filename);
```

- **Description:** Ingests and spins up background music (BGM) utilizing standard Ogg Vorbis compression formatting configured for infinite looping. Initializing a new music file stream automatically terminates and frees the memory allocation of any previously running track.
- **Arguments:**
 - `filename` : The relative system path referencing the source `.ogg` audio file.

Note: There is no command to load and prepare background music in advance. Playback begins immediately when you pass a filename to `tincle_play_music`.

```
void tincle_stop_music(void);
```

- **Description:** Instantly halts any currently playing background audio stream and safely unloads it from core memory allocations.

```
int tincle_load_sound(const char* filename);
```

- **Description:** Reads a concise sound effect (SE) clip formatted as a standard WAV or compressed Ogg Vorbis file, assigning a unique structural channel lookup ID.
- **Arguments:**
 - `filename` : The relative system path referencing the target sound asset file.
- **Return Value:** Returns the newly registered sound identifier index tracking the clip. Returns `-1` if initialization or storage routines fail.

```
void tincle_play_sound(int se_id);
```

- **Description:** Dispatches a designated sound effect instance for immediate playback. The engine natively structures an automatic multi-mixing array supporting up to 16 concurrent operational sound channels, ensuring overlapping assets do not clip or terminate early.
 - **Arguments:**
 - `se_id` : The specific sound effect target index integer originally registered via `tincle_load_sound`.
-

5.5. System Information Retrieval

```
struct SDL_Renderer* tincle_get_renderer(void);
```

- **Description:** Exposes and passes the underlying core `SDL_Renderer` structural pointer initialized by the engine environment. This allows advanced workflows or AI agents to directly invoke native SDL2 APIs (`SDL_RenderFillRect`, `SDL2_gfx` geometric primitives, etc.) to construct bespoke layouts or graphic functions directly on the frame canvas.
- **Return Value:** A functional native structural pointer referencing the active `SDL_Renderer`.

```
const char* tincle_get_version(void);
```

- **Description:** Returns a structured string literal indicating the active structural engine build and specification version number of TinCLE.

- **Return Value:** A read-only constant character pointer detailing the version tracking value (e.g., "0.01").

6. Mandatory Rules

6.1. Mandatory Coding Rules for `game.c`

When generating a `game.c` file from scratch or modifying an existing codebase, AI agents and developers **must comply 100%** with the following structural mandates:

1. **Unified Entry Point Cross-Platform Conditional Structure:** The application entry point must handle cross-platform compilation flawlessly using conditional compilation directives. Inside `game.c`, you **must strictly implement** the dual entry point architecture utilizing `#ifdef __EMSCRIPTEN__` for Web/WASM targets and `WinMain` with `<windows.h>` for native Windows targets.

Exact code template required in `game.c`:

```
#ifdef __EMSCRIPTEN__
int main(int argc, char* argv[]) {
    tincle_start(game_load, "YOUR_GAME_TITLE", true, 640, 480);
    return 0;
}
#else
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
    tincle_start(game_load, "YOUR_GAME_TITLE", true, 640, 480);
    return 0;
}
#endif
```

For the Windows native build path, you **must include** `<windows.h>` at the top of the file (wrapped cleanly within the non-emscripten block) to ensure flawless native `-mwindows` compilation.

2. **Framerate Independence (Delta Time Multiplication):** All spatial movements, animations, physics, and custom timer calculations **must without exception** be multiplied by `float dt` (the elapsed frame time in seconds) passed down into the `update` callback.
3. **Hybrid Keyboard/Gamepad Movement Logic:** When implementing spatial locomotion routines, it is highly recommended to design a modern hybrid processing layout: establish the gamepad analog stick tilt (`tincle_pad_axis_x / y`) as the baseline input vector, and immediately overwrite/synthesize it with maximum constants (`1.0f / -1.0f`) if keyboard or pad directional keys are currently engaged.
4. **Absolute Asset Loading Pre-requisite:** Within the `game_load` procedure, you **must explicitly invoke** `tincle_load_font("font.ttf");` **as the absolute first operational line**. Failing to initialize this default font handle will cause an immediate engine fatal crash during any downstream text rendering routines. Consequently, the minimum asset architecture of any distribution must contain at least one valid `.ttf` file.
5. **Conditional Use of Native SDL APIs:** If advanced routines require direct access to SDL operational structures (such as scancodes), you must conditionally switch the header paths using `#ifdef __EMSCRIPTEN__` to ensure cross-platform compatibility. For Windows native builds, include headers from the local directory using double quotes (e.g., `#include "libs/SDL.h"` or `#include "libs/SDL_scancode.h"`). For Emscripten Web builds, you must include the Emscripten-bundled SDK headers using angle brackets (e.g., `#include <SDL2/SDL.h>`). However, **you must never write code that directly manipulates or alters the core SDL window handles, active renderers, or primary event processing systems**. All application-level game logic and rendering flows must be fully abstracted and wrapped within native TinCLE APIs.
6. **Mandatory State Registration inside `game_load`:**
 - You **MUST** call `tincle_add_state()` exclusively **WITHIN** the `game_load()` function.

- DO NOT invoke `tincle_add_state()` inside `main()` or `WinMain()`. The internal engine architecture initializes the state handles within `tincle_start()`, meaning any downstream states registered before `tincle_start()` will be completely wiped out, resulting in a fatal black screen.

6.2. Mandatory AI Coding Constraints & Rules

To ensure 100% compilation on the first attempt without compilation errors, the AI must strictly adhere to these 8 coding rules:

1. **NO SEARCH CITATIONS IN CODE:** Never include text-based source citations, inline footnotes, or reference annotations (such as `[cite: 1]`, `[1]`, or `(Source: ...)`) anywhere inside or outside the code block. These annotations instantly cause syntax errors in C. Output ONLY valid, clean C source code.
2. **Absolute Pure C (C99 Standard):** Do not mix C++ syntax (e.g., no classes, no namespaces).
3. **Double Quotes & Platform-Specific Headers:** Always include engine core headers using `#include "libs/tincle.h"`. When including raw SDL headers, you must cleanly branch your code using `#ifdef __EMSCRIPTEN__` — use `#include "libs/SDL.h"` or `#include "libs/SDL2_gfxPrimitives.h"` or etc. for Windows native and `#include <SDL2/SDL.h>` or `#include <SDL2/SDL2_gfxPrimitives.h>` or etc. for Emscripten Web targets.
4. **Standard Library Inclusions:** If you utilize mathematical operations (such as `sinf`, `cosf`, `sqrtf`, `fabsf`), you must explicitly `#include <math.h>`. Do not assume standard headers are pre-included.
5. **Mandatory Font Preloading:** The framework requires an operational font to initialize text routines. You must execute `tincle_load_font("font.ttf");` immediately at the top of the `game_load()` lifecycle sequence before loading other images or sounds.
6. **Strict State-Separation Model:** Do not write a chaotic single-loop game block. Split game states cleanly (e.g., Title Screen, Gameplay Loop, Game Over Screen) using the built-in State Management API.
7. **Always Use Approved Build Tools (`tinclec` & `emcass`):** When guiding the user on how to build the game, instruct them based on the target platform. Instruct them to execute the bundled `tinclec` tool for native Windows desktop builds, and the `emcass` tool for Web/WASM compilation. Never instruct them to use raw `tcc` or raw `emcc` commands directly.
8. **Never Confuse Asset Utilities:**
 - `asspack.exe` is used to pack all game media assets (images, fonts, sounds) into a secure `game.data` package.
 - `embedres.exe` is used ONLY for embedding binary metadata (like `icon.ico` and application manifest strings) directly into the executable file header. It cannot process game media assets.

7. Creation of Release Version (Distribution Instructions)

When ready to build the final production release for distribution, follow this strict five-step pipeline:

- **Asset Credit Management:** If you recognize, extract, or utilize third-party assets from a provided ZIP file or external sources, you must automatically append the appropriate credit and attribution information to `CREDITS.txt`. Do not leave `CREDITS.txt` as the default template if new third-party assets are introduced.

Step 1: Hide the Debug Console

Locate the `tincle_start` call inside the conditional entry points (`main` for Web or `WinMain` for Windows) in `game.c`. Change the third argument (`is_debug`) from `true` to `false`.

```
// Example: Hiding the runtime debug log or console output
tincle_start(game_load, "My Ultimate Game", false, 640, 480);
// Set this to false for production!
```

Step 2: High-Density Asset Packing via `asspack` (for Windows)

To bundle all external textures, audio clips, and font files into an uncrackable, optimized unified package for Windows Desktop deployment, execute the `asspack` command in the terminal. This creates a single consolidated virtual file called `game.data`.

```
> asspack font.ttf player.png enemy.png bgm.ogg shoot.wav
```

Note: Once `game.data` is generated, the Windows engine will automatically look inside this packed binary container. However, if a raw asset file exists locally, it takes precedence over the packed asset. When distributing your game, pack all assets into `game.data` and do not include raw asset files in the release archive.

Step 3: Web/WASM Production Build via `emcass` (for Web)

For web browser deployment, the asset packaging and compilation are unified into a single command using `emcass`. You do not need to use `asspack` manually for Web builds; `emcass` handles both asset preloading and generation of the web deliverables automatically.

```
emcass index.html "My Ultimate Game" main.c player.c stage.c
```

Note: This command scans your local development directory, compiles the code via the Emscripten toolchain, packs all detected assets into `index.data`, and uses `template.html` to output the complete browser-ready package with the mandatory fixed base name `index`. Requirements: When executing a Web build, ensure that `CREDITS.txt` is placed in the exact same folder as the source code before running the build. This is strict because `emcass` requires `CREDITS.txt` in that directory to display credits properly; otherwise, the web-version credit screen will be blank.

Step 4: Final Deployment Package Checklist (Windows Desktop Target)

Distribute the final Windows desktop build by packaging the following **"8 structural items"** inside a ZIP archive. Do not include individual raw source assets, header files, or `.c` files in the player package:

1. `game.exe` (The compiled game binary generated by `tinclec` - can be renamed)
2. `game.data` (The packed resource binary containing all assets generated by `asspack.exe`)
3. `tincle.dll` (The TinCLE core engine runtime)
4. `SDL2.dll` (Official SDL2 library)
5. `SDL2_ttf.dll` (Official SDL_ttf library)
6. `SDL2_gfx.dll` (Official SDL2_gfx library)
7. `CREDITS.txt` (License documentation for the engine and libraries)
8. `readme.txt` (The instruction manual file for the game itself)

Note: Create and prepare a `readme.txt` file containing a description of the game to be distributed.

Step 5: Final Deployment Package Checklist (Web Browser Target)

Distribute the final Web browser build by packaging the following **"5 structural items"** generated by `emcass` into a single ZIP archive for platform deployment (e.g., itch.io HTML5 project upload):

1. `index.html` (The web page housing the game canvas - MUST NOT be renamed)
2. `index.js` (The JavaScript glue-code generated by Emscripten)
3. `index.wasm` (The compiled WebAssembly binary core)
4. `index.data` (The virtual filesystem package containing preloaded assets)
5. `CREDITS.txt` (License documentation for the engine, Emscripten components, and libraries)

MANDATORY INSTRUCTIONS FOR `readme.txt` CREATION

When the AI agent or developer prepares the `readme.txt` (or platform game page description), the following specifications must be explicitly detailed without exception:

1. Input Shortcuts & Window Controls Disclosure:

- **For Windows Desktop Players:** Clearly state that pressing `Alt + Enter` toggles between Fullscreen and Windowed mode, and pressing `Alt + Backspace` immediately resets the game window to its default resolution scale. (Since these core shortcuts are invisible, players have no way of knowing them unless documented here).
- **For Web Browser Players:** Clearly state that players can toggle Fullscreen mode by clicking the dedicated HTML canvas fullscreen button provided on the game page.

2. Mandatory Game Creator Copyright Attribution:

- The `readme.txt` must clearly display the user's/developer's own copyright notice as the primary game creator (e.g., Copyright (C) 2026 [Your Name / Team Name]).
- **CRITICAL AI WARNING:** When creating code or release readmes, do NOT just only append the TinCLE framework copyright notice (Copyright (C) 2026 michyo). While the engine license requires credit attribution to the engine author, the game itself belongs to the game creator. The AI must explicitly separate the Game Creator Copyright from the Engine Framework Credits and ensure the game author's identity is clearly defined.

8. Standard Implementation Boilerplate Code for `game.c`

This section provides the comprehensive, bulletproof source code boilerplate template that AI agents must utilize as the structural foundation when spinning up a brand new game project from scratch.

```
#define SDL_MAIN_HANDLED
#include "libs/tincle.h"
// [TCC WORKAROUND] Prevent 'M_PI redefined' warning from SDL2_gfx
#undef M_PI
// Automatically switch between headers for the Web version and the Windows version
#ifdef __EMSCRIPTEN__
    #include <SDL2/SDL2_gfxPrimitives.h> // The web version uses Emscripten's built-in gfx
#else
    #include "libs/SDL2_gfxPrimitives.h" // The Windows version uses the gfx files in the lib folder
    #include <windows.h> // Required for native Windows structures
#endif
#include <stdio.h>
#include <math.h>
// [TCC WORKAROUND] Fix 'undefined symbol sinf' error in Windows TCC environments.
// Maps the C99 float function 'sinf' to the standard double 'sin' via compiler macro.
#ifndef sinf
#define sinf(x) ((float)sin((double)(x)))
#endif
// Global System Variables & Options
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; // Used for sine wave animations
static int g_img_player = -1;
// =====
// State: Title Screen (State ID: 0)
// =====
void title_update(float dt) {
    // Advance to Gameplay State via Space key or Gamepad Start button
    if (tincle_key_pressed(TINCLE_KEY_SPACE) || tincle_pad_pressed(TINCLE_PAD_START)) {
```

```

    tincle_change_state(1);
}
}

void title_draw(void) {
    tincle_clear(TINCLE_COLOR_BLACK);
    // MANDATORY: Always use tincle_draw_text_center for perfect UI text alignment!
    tincle_draw_text_center("YOUR_GAME_TITLE", 320, 180, TINCLE_FONT_L, TINCLE_COLOR_WHITE);
    tincle_draw_text_center("PRESS SPACE TO START", 320, 300, TINCLE_FONT_M, TINCLE_COLOR_GREEN);
}

// =====
// State: Gameplay Loop (State ID: 1)
// =====

void play_update(float dt) {
    // Accumulate delta time for dynamic calculations
    g_game_time += dt;
    // Framerate Independence: Speed is always multiplied by delta time (dt)
    float move_speed = 200.0f;
    if (tincle_key_pressed(TINCLE_KEY_LEFT) || tincle_pad_pressed(TINCLE_PAD_LEFT)) g_player_x -= move_speed * dt;
    if (tincle_key_pressed(TINCLE_KEY_RIGHT) || tincle_pad_pressed(TINCLE_PAD_RIGHT)) g_player_x += move_speed * dt;
    if (tincle_key_pressed(TINCLE_KEY_UP) || tincle_pad_pressed(TINCLE_PAD_UP)) g_player_y -= move_speed * dt;
    if (tincle_key_pressed(TINCLE_KEY_DOWN) || tincle_pad_pressed(TINCLE_PAD_DOWN)) g_player_y += move_speed * dt;
    // Returning to the Title Screen sequence
    if (tincle_key_pressed(TINCLE_KEY_ESCAPE) || tincle_pad_pressed(TINCLE_PAD_BACK)) {
        tincle_change_state(0);
    }
}

void play_draw(void) {
    // Clear display background using deep navy color parameters
    tincle_clear((TincLE_Color){20, 20, 50, 255});
    tincle_draw_text("GAME PLAYING", 20, 20, TINCLE_FONT_M, TINCLE_COLOR_GREEN);
    // Render player asset dynamically
    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_FONT_M, TINCLE_COLOR_RED);
    }
    // [EXAMPLE 1] Render a filled square using native SDL API
    SDL_Rect rect = { 100, 100, 50, 50 };
    SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);
    SDL_RenderFillRect(renderer, &rect);
    // [EXAMPLE 2] Render a floating alpha ellipse using SDL2_gfx primitives
    filledEllipseRGBA(renderer, 400, 300, 50, 25, 255, 0, 0, 128);
    // [EXAMPLE 3] Render a filled polygon triangle using SDL2_gfx filledTrigonRGBA.
    // This demonstrates how to construct a triangle instead of utilizing a non-existent "triangleRGBA" function.
    // The top vertex floats up and down smoothly via sinf() from math.h!
    int tri_top_y = 150 + (int)(sinf(g_game_time * 5.0f) * 15.0f);
    filledTrigonRGBA(renderer,
        500, tri_top_y, // Vertex 1 (Top, moving dynamically)
        460, 200,      // Vertex 2 (Bottom Left)
        540, 200,      // Vertex 3 (Bottom Right)
        0, 255, 255, 200 // Color: Cyan with Alpha
    );
}

// =====
// System Lifecycle: Initialization & Asset Preloading
// =====

void game_load(void) {
    // Cache the underlying core SDL_Renderer context to enable native graphics extensions
    renderer = tincle_get_renderer();
    // MANDATORY MANDATE: Font asset initialization must be executed first to avoid fatal engine crashes
    tincle_load_font("font.ttf");
    // Optional downstream asset loading queues
    g_img_player = tincle_load_image("player.png");
    // Bind and register the operational lifecycle callback procedures to their respective scene handles
    tincle_add_state(0, title_update, title_draw);

```

```
tincle_add_state(1, play_update, play_draw);
// CRITICAL MANDATE: You must switch to your starting state at the end of game_load!
tincle_change_state(0);
}

// =====
// Main Entry Point
// =====

#ifdef __EMSCRIPTEN__
// Standard entry for Web (Emscripten) mode
int main(int argc, char* argv[]) {
    tincle_start(game_load, "YOUR_GAME_TITLE", true, 640, 480);
    return 0;
}
#else
// WinMain entry for Windows (TCC, etc.) mode
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
    tincle_start(game_load, "YOUR_GAME_TITLE", true, 640, 480);
    return 0;
}
#endif
```