Basic オーディオ再生

Basic で microSD カードのオーディオファイルを再生するサンプルプログラム。

cpp

```cpp
#include <M5Unified.h>
#include <SPI.h>
#include <SD.h>

#define SD_SPI_CS_PIN   4
#define SD_SPI_SCK_PIN  18
#define SD_SPI_MISO_PIN 19
#define SD_SPI_MOSI_PIN 23

bool playWAVFromSD(const char* filename, uint32_t repeat = 1, int channel = -1, bool stop_current =
bool playWAVMemory(File& wavFile, size_t fileSize, uint32_t repeat, int channel, bool stop_current)
bool playWAVSegmented(const char* filename, uint32_t repeat, int channel, bool stop_current);
void printf_log(const char *format, ...);
void println_log(const char *str);

void setup() {
    M5.begin();
    Serial.begin(115200);
    M5.Display.fillRect(0, 0, 320, 240, WHITE);
    M5.Display.setTextColor(BLACK);
    M5.Display.setFont(&fonts::FreeMonoBold9pt7b);
    M5.Display.setCursor(0, 0);

    // SD Card Initialization
    SPI.begin(SD_SPI_SCK_PIN, SD_SPI_MISO_PIN, SD_SPI_MOSI_PIN, SD_SPI_CS_PIN);

    if (!SD.begin(SD_SPI_CS_PIN, SPI, 25000000)) {
        println_log("Card failed, or not present");
        while (1) delay(1000);
    }

    uint64_t cardSize = SD.cardSize() / (1024 * 1024);
    printf_log("SD Card Size: %lluMB\n", cardSize);

    println_log("Starting audio playback...");
    playWAVFromSD("/sample-6s.wav", 1, -1, true);
}

void loop() {

}

bool playWAVFromSD(const char* filename, uint32_t repeat, int channel, bool stop_current) {
    if (!SD.exists(filename)) {
        println_log("File does not exist!");
        return false;
    }

    File wavFile = SD.open(filename, FILE_READ);
    if (!wavFile) {
```

```
        println_log("Failed to open file!");
        return false;
    }

    size_t fileSize = wavFile.size();
    printf_log("File size: %d Byte (%.2f KB)\n", fileSize, fileSize/1024.0);

    size_t freeHeap = ESP.getFreeHeap();
    printf_log("Free heap: %d bytes\n", freeHeap);

    if (fileSize < freeHeap / 2) {
        return playWAVMemory(wavFile, fileSize, repeat, channel, stop_current);
    }
// ファイルが大きすぎるため、分割再生を使用
    wavFile.close();
    return playWAVSegmented(filename, repeat, channel, stop_current);
}

bool playWAVMemory(File& wavFile, size_t fileSize, uint32_t repeat, int channel, bool stop_current)
    uint8_t* wavData = (uint8_t*)malloc(fileSize);
    if (!wavData) {
        println_log("Memory allocation failed!");
        wavFile.close();
        return false;
    }

    println_log("Loading file to memory...");
    size_t bytesRead = wavFile.read(wavData, fileSize);
    wavFile.close();

    if (bytesRead != fileSize) {
        printf_log("Read error: %d/%d bytes\n", bytesRead, fileSize);
        free(wavData);
        return false;
    }

    println_log("Starting playback...");
    bool result = M5.Speaker.playWav(wavData, fileSize, repeat, channel, stop_current);

    if (result) {
        while (M5.Speaker.isPlaying()) {
            delay(100);
        }
        println_log("Playback completed!");
    }

    free(wavData);
    return result;
}

// 分割再生（大きなファイル） - メモリ使用の最適化
bool playWAVSegmented(const char* filename, uint32_t repeat, int channel, bool stop_current) {
    File wavFile = SD.open(filename, FILE_READ);
```

```
    if (!wavFile) return false;

    uint8_t header[44];
    if (wavFile.read(header, 44) != 44) {
        wavFile.close();
        return false;
    }

    if (strncmp((char*)header, "RIFF", 4) != 0 ||
        strncmp((char*)header + 8, "WAVE", 4) != 0) {
        println_log("Invalid WAV format");
        wavFile.close();
        return false;
    }

    uint32_t totalFileSize = *(uint32_t*)(header + 4) + 8;
    uint32_t sampleRate = *(uint32_t*)(header + 24);
    uint16_t channels = *(uint16_t*)(header + 22);
    uint16_t bitsPerSample = *(uint16_t*)(header + 34);

    printf_log("WAV: %dHz, %dch, %dbit\n", sampleRate, channels, bitsPerSample);

    size_t dataSize = totalFileSize - 44;

    size_t bytesPerSample = (bitsPerSample / 8) * channels;
    size_t chunkSizeInSamples = 16384 / bytesPerSample;
    size_t chunkSize = chunkSizeInSamples * bytesPerSample;

    printf_log("Chunk size: %d bytes (%d samples)\n", chunkSize, chunkSizeInSamples);

    uint8_t* chunkBuffer = nullptr;
    size_t actualChunkSize = chunkSize;

    while (actualChunkSize >= 4096 && !chunkBuffer) {  // 最小4KB
        chunkBuffer = (uint8_t*)malloc(actualChunkSize + 44);
        if (!chunkBuffer) {
            actualChunkSize /= 2;
            chunkSizeInSamples = actualChunkSize / bytesPerSample;
            actualChunkSize = chunkSizeInSamples * bytesPerSample;
            printf_log("Retrying with smaller chunk: %d bytes\n", actualChunkSize);
        }
    }

    if (!chunkBuffer) {
        println_log("Buffer allocation failed even with small chunks!");
        wavFile.close();
        return false;
    }

    printf_log("Using chunk size: %d bytes\n", actualChunkSize);
    println_log("Starting segmented playback...");

    for (uint32_t rep = 0; rep < repeat; rep++) {
```

```
        size_t totalRead = 0;
        int segmentNum = 0;

        wavFile.seek(44);

        while (totalRead < dataSize) {
            size_t bytesToRead = min(actualChunkSize, dataSize - totalRead);

            memcpy(chunkBuffer, header, 44);
            uint32_t chunkFileSize = bytesToRead + 36;
            memcpy(chunkBuffer + 4, &chunkFileSize, 4);
            memcpy(chunkBuffer + 40, &bytesToRead, 4);

            size_t bytesRead = wavFile.read(chunkBuffer + 44, bytesToRead);
            if (bytesRead == 0) break;

            totalRead += bytesRead;
            segmentNum++;

            if (segmentNum % 5 == 1) {
                printf_log("Segment %d (%.1f%%)\n",
                           segmentNum, (float)totalRead / dataSize * 100.0);
            }

            bool playResult = M5.Speaker.playWav(chunkBuffer, bytesRead + 44, 1, channel, stop_curr

            if (!playResult) {
                printf_log("Segment %d failed\n", segmentNum);
                break;
            }

            while (M5.Speaker.isPlaying()) {
                delay(5);
            }

            delay(10);
        }

        if (rep < repeat - 1) {
            delay(1000);
        }
    }

    free(chunkBuffer);
    wavFile.close();
    println_log("Segmented playback completed!");
    return true;
}

void printf_log(const char *format, ...) {
    char buf[256];
    va_list args;
    va_start(args, format);
```

```
    vsnprintf(buf, 256, format, args);
    va_end(args);
    Serial.print(buf);
    M5.Display.printf(buf);
}

void println_log(const char *str) {
    Serial.println(str);
    M5.Display.println(str);
}
```