

DCC192

2025/1

UF  G

Desenvolvimento de Jogos Digitais

A16: Áudio II – Gravação/Reprodução

Prof. Lucas N. Ferreira

Plano de aula



- ▶ Gravação de Áudio em Video Games
- ▶ Carregando e Reproduzindo Áudio em SDL
- ▶ Problemas de Gerenciamento de Áudio em Jogos
- ▶ Sistema de Gerenciamento de Áudio
 - ▶ Play, Pause, Resume e Stop
 - ▶ Lidando com Repetições
 - ▶ Efeitos em Processamento de Sinais
 - ▶ Pitch Shift

Quantos sons diferentes nesse trecho do jogo?

m



- ▶ Música de fundo
- ▶ Andar água
- ▶ Andar grama
- ▶ ataque
- ▶ Bomba
- ▶ Menu
- ▶ Seleção
- ▶ Dano
- ▶ ...

Gravação de Áudio em Video Games

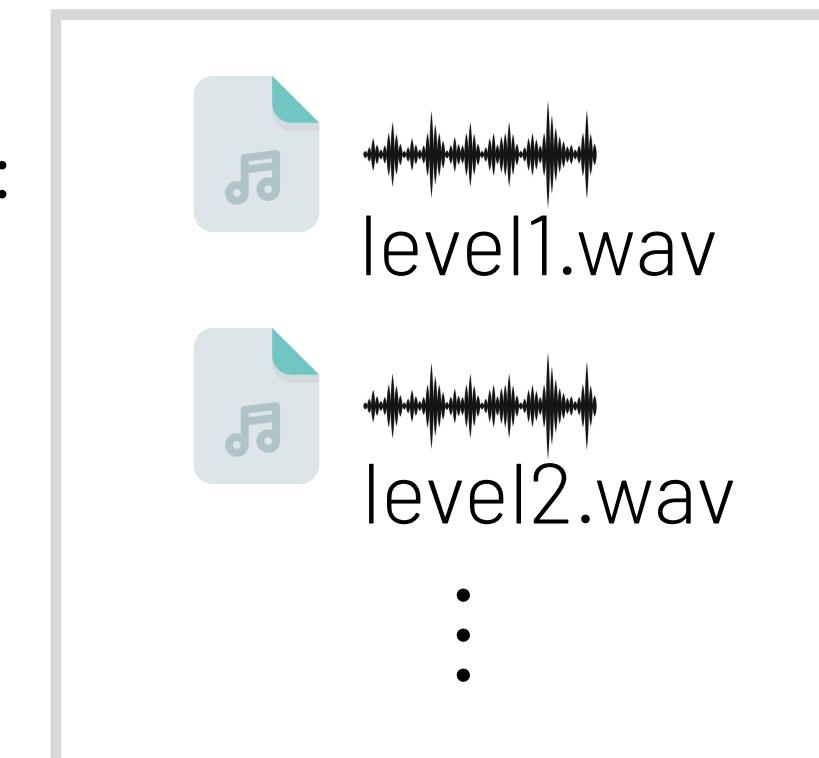


Atualmente, além de áudios sintetizados em tempo real, é muito comum os jogos utilizarem áudios pré-gravados, que são carregados pelo jogo e reproduzidos no momento adequado.



Efeitos
Sonoros:

Trilas
Sonoras:



```
void Initialize() {  
    ...  
    j = LoadSound(jump.wav)  
    ...  
}  
  
void Input() {  
    ...  
    if(keyPressed(Up))  
        PlaySound(j)  
    ...  
}
```

Carregando e Reproduzindo Áudio em SDL



Para carregar arquivos de áudio em SDL, precisamos usar uma biblioteca adicional `SDL_Mixer.h`

```
// Inicializa SDL_mixer com suporte a MP3, OGG, etc.  
Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 2048);  
  
// Altera o número de canais (ex. 3). Se esse função não for chamada, a SDL_Mixer cria 8 canais por padrão.  
Mix_AllocateChannels(3);  
  
// Carrega o arquivo de áudio (pode ser WAV, MP3, OGG...)  
Mix_Chunk* chunk = Mix_LoadWav("audio.wav");  
  
// Toca o som no canal especificado no primeiro canal disponível (-1), sem loop (0)  
Mix_PlayChannel(-1, som, 0);  
  
// Verifica se um som está tocando no canal especificado (ex. 0): 1 se estiver, 0 caso contrário  
Mix_Playing(0);  
  
// Para o som no canal especificado (ex. 0)  
Mix_HaltChannel(0);  
  
// Libera recursos  
Mix_FreeMusic(musica);  
Mix_CloseAudio();
```

SDL Mixer: Canais Não, Trilhas

Os canais da SDL_Mixer **não são os canais mono/stereo**, mas sim **trilhas** de áudio que podem ser tocadas ao mesmo tempo. Por exemplo, se você estiver usando 3 canais:



Canal 0:	level1.wav
Canal 1:	walk.wav
Canal 2:	sword.wav

Você poderá tocar 3 sons diferentes simultâneamente.

Problemas de Gerenciamento de Áudio em Jogos



Considere um sistema com 3 canais e a seguinte situação durante o jogo:

1. `Mix_PlayChannel(-1, music)` // Tocar musica → seleciona canal 0
2. Considere que a música terminou normalmente
3. `Mix_PlayChannel(-1, sword)` // Tocar efeito da espada → seleciona canal 0 novamente!
4. Considere que o jogador apertou pause no jogo e queremos parar a música
5. `Mix_HaltChannel(0)`

O que acabou de acontecer?

Nós acidentalmente paramos o som da espada!

Problemas de Gerenciamento de Áudio em Jogos



Considere um sistema com 3 canais, mas que 4 áudios foram requisitados ao mesmo tempo:

1. `Mix_PlayChannel(-1, music)` // Tocar musica → seleciona canal 0

`Mix_PlayChannel(-1, walk)` // Tocar efeito sonoro de andar → seleciona canal 1

`Mix_PlayChannel(-1, sword)` // Tocar efeito sonoro de ataque → seleciona canal 2

`Mix_PlayChannel(-1, bush)` // Tocar efeito sonoro do arbusto → ?

O que irá acontecer?

O comportamento padrão da SDL_Mixer é substituir o som mais antigo, ou seja, a música!

Precisamos gerenciar os canais com mais cuidado!

Sistema de Gerenciamento de Áudio



Um sistema de gerenciamento de áudio em geral possui as seguintes funções:

```
class AudioSystem {
public:
    AudioSystem(int numChannels = 8); // Criar um sistema de áudio com um dado número de trilhas
    ~AudioSystem(); // Destruir um sistema de som

    void Update(float deltaTime); // Atualiza estado dos sons ativos

    SoundHandle PlaySound(const std::string& soundName, bool looping = false);
    void PauseSound(SoundHandle sound); // Pausar um som se ele estiver tocando no momento
    void ResumeSound(SoundHandle sound); // Retomar um som se ele estiver pausado no momento
    void StopSound(SoundHandle sound); // Parar um som se ele estiver tocando no momento
    void StopAllSounds(); // Parar todos os sons em todas as trilhas

    void CacheSound(const std::string& soundName);
private:
    struct Mix_Chunk* GetSound(const std::string& soundName);

    SoundHandle mLastHandle;
    std::vector<SoundHandle> mChannels;
    std::map<SoundHandle, HandleInfo> mHandleMap;
    std::unordered_map<std::string, Mix_Chunk*> mSounds;
};
```

Exemplo de uso

Na inicialização do jogo (classe `Game`) iremos construir um sistema de áudio:

```
bool Game::Initialize() {  
    ...  
    mAudio = new AudioSystem();  
    ...  
}
```

Para tocar um novo som, basta chamar uma função `PlaySound("sound.wav")`:

```
bool Player::Update() {  
    ...  
    mGame->GetAudioSystem()->PlaySound("sword.wav");  
    ...  
}
```

SoundHandle: Identificador de Sons



As operações com som utilizarão um **ID** único em vez de diretamente um canal **SDL_mixer**:

```
typedef unsigned int SoundHandle;

SoundHandle PlaySound(const std::string& soundName, bool looping = false);
void PauseSound(SoundHandle sound); // Pausar um som se ele estiver tocando no momento
void ResumeSound(SoundHandle sound); // Retomar um som se ele estiver pausado no momento
void StopSound(SoundHandle sound); // Parar um som se ele estiver tocando no momento
```

1. Quando reproduzimos um som com **PlaySound()**, ele recebe um **ID** único;
 - ▶ Chamaremos esse **ID** de **SoundHandle**, um inteiro sem sinal (**unsigned int**)
 - ▶ Um **SoundHandle** pode representar até $2^{32} \sim 4$ bilhões de identificadores únicos!
2. Quando quisermos dar **Pause()**, **Resume()** ou **Stop()** em um som alvo, utilizaremos um **SoundHandle** para especificá-lo.

HandleInfo: Estado do Som



Para gerenciar os sons em andamento, iremos armazenar o estado de cada `SoundHandle` ativo em uma estrutura chamada `HandleInfo`:

```
struct HandleInfo {
    std::string mSoundName;
    int mChannel = -1;
    bool mIsLooping = false;
    bool mIsPaused = false;
};

std::map<SoundHandle, HandleInfo> mHandleMap;
```

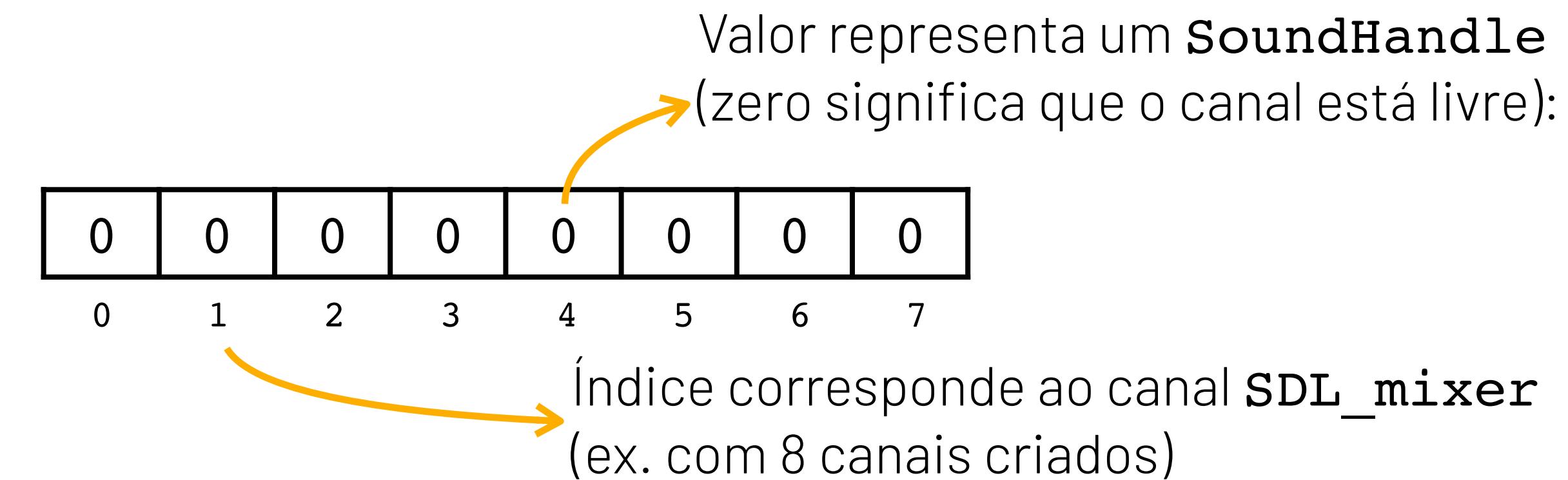
Ela nos permitirá verificar o estado dos sons em andamento:

- ▶ Quais sons estão em loop ou pausados;
- ▶ Qual o som mais antigo ou mais recente;

mChannels: Armazenando canais

Os `SoundHandles` são criados incrementalmente (de um em um) e associados a um canal da `SDL_Mixer` por meio de um vetor chamado `mChannels`:

```
SoundHandle mLastHandle;  
std::vector<SoundHandle> mChannels;
```



A `mLastHandle` é um contador de sons que repesenta o `SoundHandle` do último som tocado:

```
SoundHandle mLastHandle = 0;  
  
mLastHandle = 1 → PlaySound("level1.wav")  
mLastHandle = 2 → PlaySound("walk.wav")  
mLastHandle = 3 → PlaySound("sword.wav")  
mLastHandle = 4 → PlaySound("sword.wav")
```

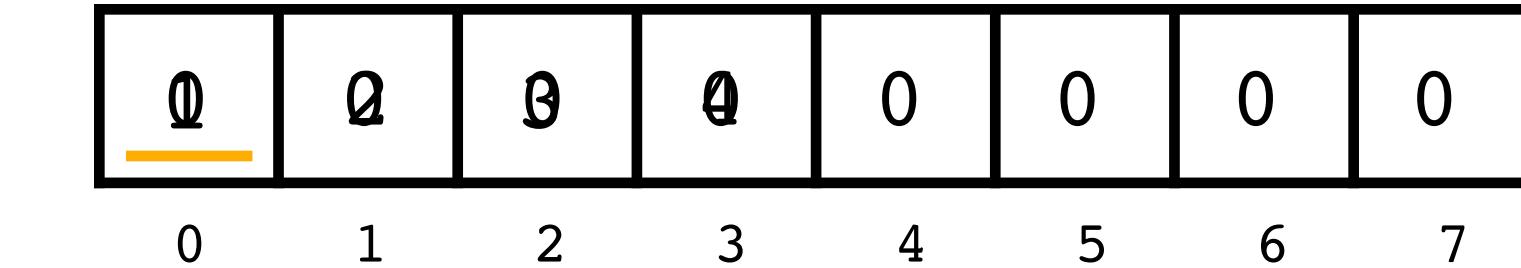
PlaySound: Reproduzindo Sons



Os `SoundHandles` são criados incrementalmente (de um em um) e associados a um canal da `SDL_Mixer` por meio de um vetor chamado `mChannels`:

```
SoundHandle mLastHandle = 0
```

```
vector<SoundHandle> mChannels
```



```
PlaySound("level1.wav")
```

```
PlaySound("walk.wav")
```

```
PlaySound("sword.wav")
```

```
PlaySound("sword.wav")
```

```
map<SoundHandle, HandleInfo>  
mHandleMap;
```

"lev"	"wal"	"swo"	"swo"
True	False	False	False
False	False	False	False

1 2 3 4

1. Percorrer o vetor `mChannels` até que (`mChannels[i] == 0`) → canal `i` está disponível
2. Se houver um canal disponível:
 - ▶ Incrementar (`mLastHandle++`) e atribuir ao canal `i` encontrado (`mChannels[i] = mLastHandle`)
 - ▶ Tocar o som usando `SDL_MixPlayChannel(i)` e adicionar um novo `HandleInfo` a `mHandleMap`;

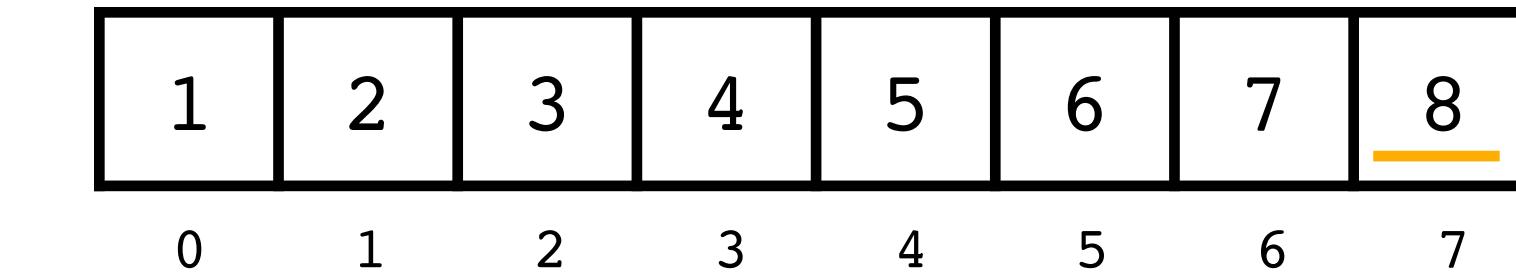
PlaySound: Reproduzindo Sons



Os `SoundHandles` são criados incrementalmente (de um em um) e associados a um canal da `SDL_Mixer` por meio de um vetor chamado `mChannels`:

```
SoundHandle mLastHandle = 8
```

```
vector<SoundHandle> mChannels
```



```
PlaySound("level1.wav")
```

```
PlaySound("sword.wav")
```

```
...
```

```
PlaySound("sword.wav")
```

```
map<SoundHandle, HandleInfo>  
mHandleMap;
```

"lev"	"wal"	"swo"	"swo"	"swo"	"swo"	"swo"	"swo"
True	False						
False							
1	2	3	4	5	6	7	8

3. Se nenhum canal estiver disponível, temos que escolher algum para sobrescrever:
 - i. Se houver instâncias existentes do som (ex. "sword.wav"), substitua a instância mais antiga desse mesmo som;
 - ii. Caso contrário, substitua o som não em loop mais antigo;
 - iii. Caso contrário, substitua o som mais antigo.

Update: Atualizando estado de sons ativos



A cada quadro, na função `AudioSystem::Update(deltaTime)`, vamos verificar se um som ativo não está mais tocando, para reinicializar seu

```
SoundHandle mLastHandle = 4
```

```
PlaySound("level1.wav")
```

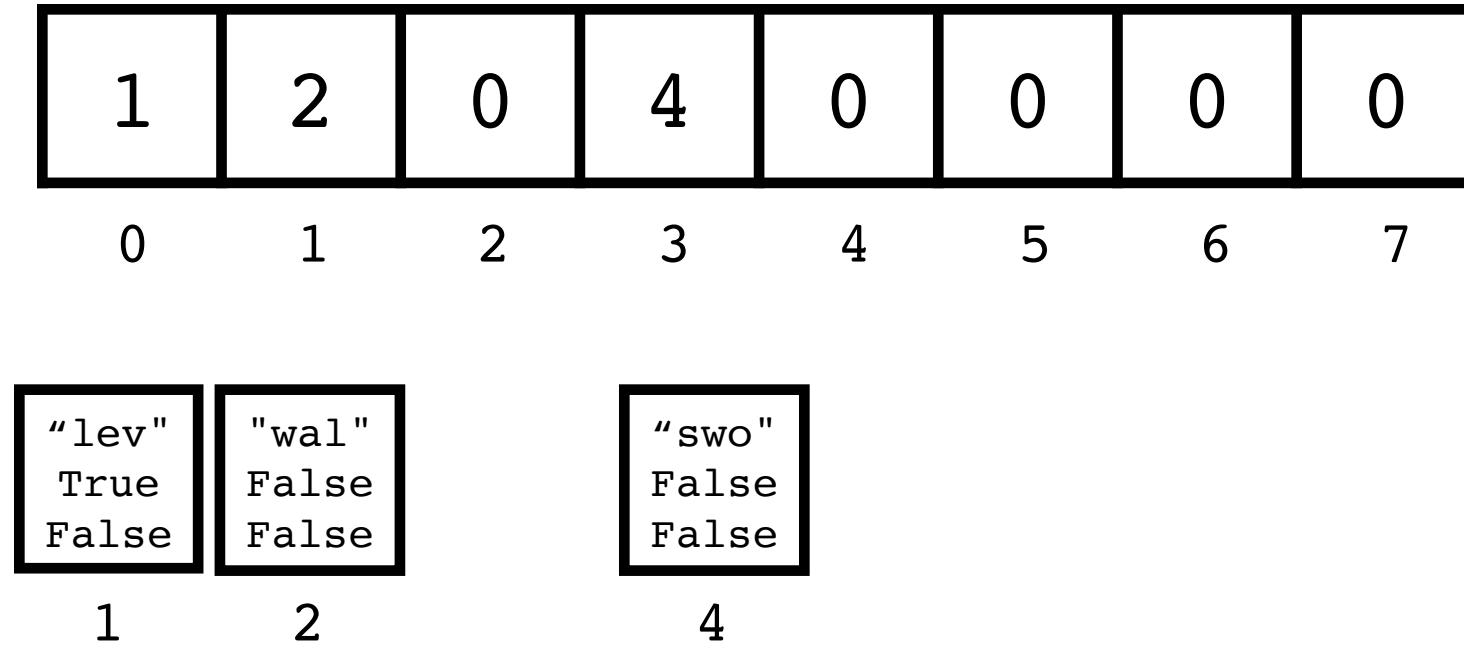
```
PlaySound("walk.wav")
```

```
PlaySound("sword.wav")
```

```
PlaySound("sword.wav")
```

```
vector<SoundHandle> mChannels
```

```
map<SoundHandle, HandleInfo>  
mHandleMap;
```



1. Percorrer o vetor `mChannels`:
2. Se o canal estiver ativo (`mChannels[i] != 0`) e não estiver mais tocando (`Mix_Playing(i) == 0`)
 - i. Liberar o canal (`mChannels[i] = 0`)
 - ii. Remover o `HandleInfo` desse som do mapa de `SoundHandles` (`mHandleMap.erase(mChannels[i])`)

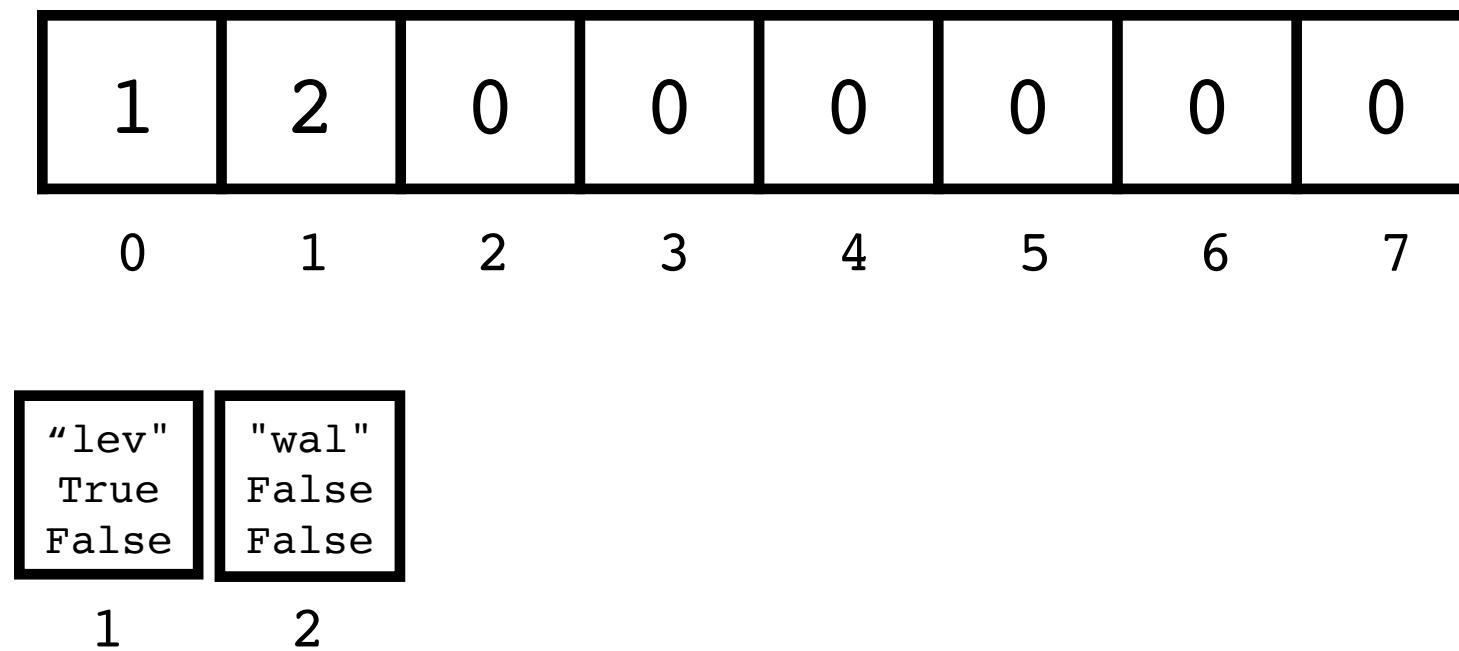
Pause e Stop



Para Pausar/Parar um som, basta (1) verificar se o SoundHandle existe no mHandleMap e (2) se ele já não está pausado/parado:

```
SoundHandle mLastHandle = 2           vector<SoundHandle> mChannels  
  
PlaySound("level1.wav")  
PlaySound("walk.wav")  
PauseSound(1)  
StopSound(0)
```

map<SoundHandle, HandleInfo>
mHandleMap;

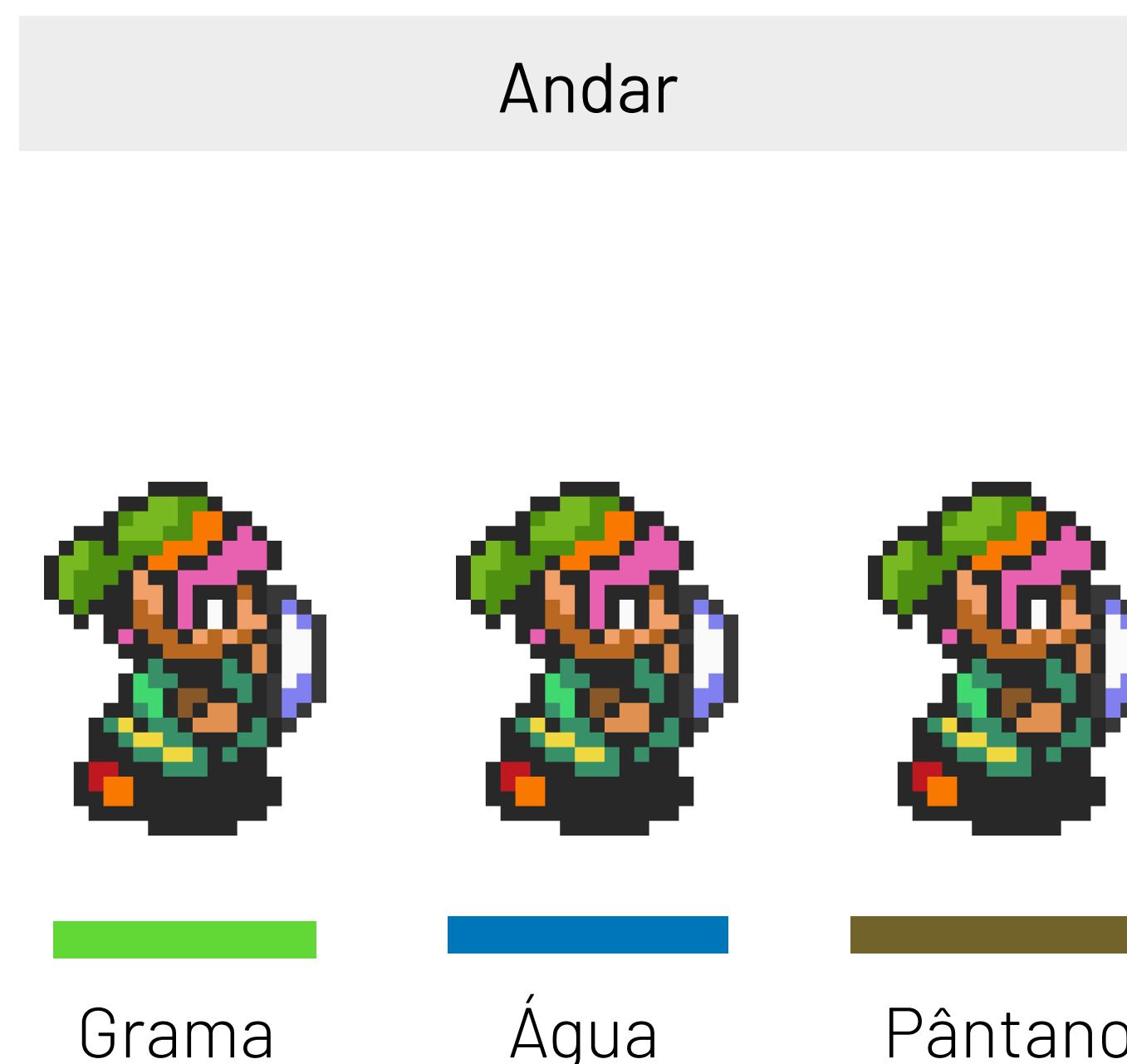


1. Se o **SoundHandle** existir e não estiver pausado/parado:
 - i. Recuperar o canal associado a esse **SoundHandle** (`canal = mHandleMap[sound].mChannel`)
 - ii. Pausar/Parar esse som: `Mix_Pause(canal)`/`Mix_HaltChannel(canal)`
No caso de parar, também precisa liberar o canal e remover a entrada do **mHandleMap**, como fizemos no **Update**

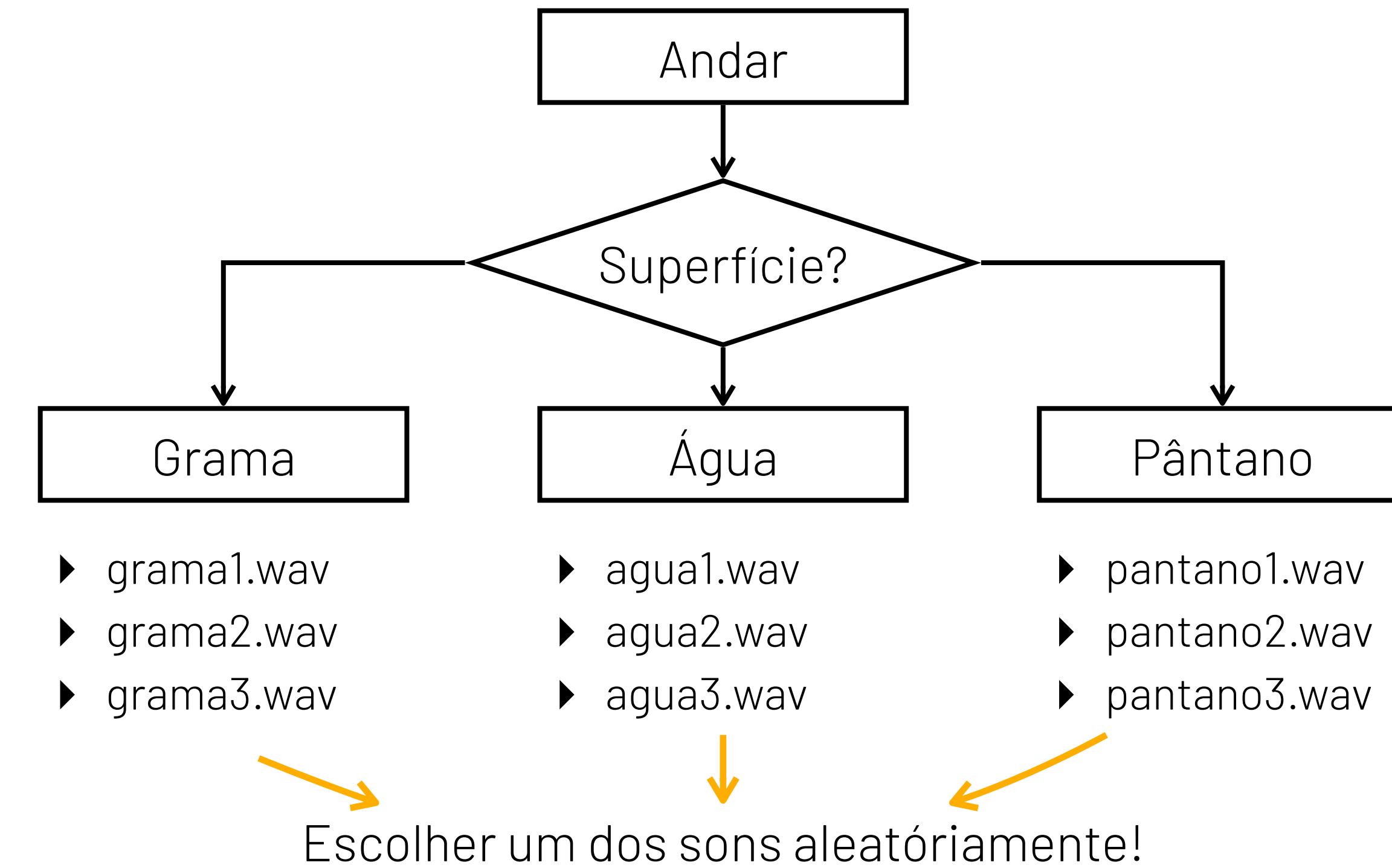
Lidando com Repetições



Quando reproduzimos o mesmo efeito de som repetidas vezes, como é o caso de animações recorrentes no jogo, ele se torna rapidamente enjoativo.



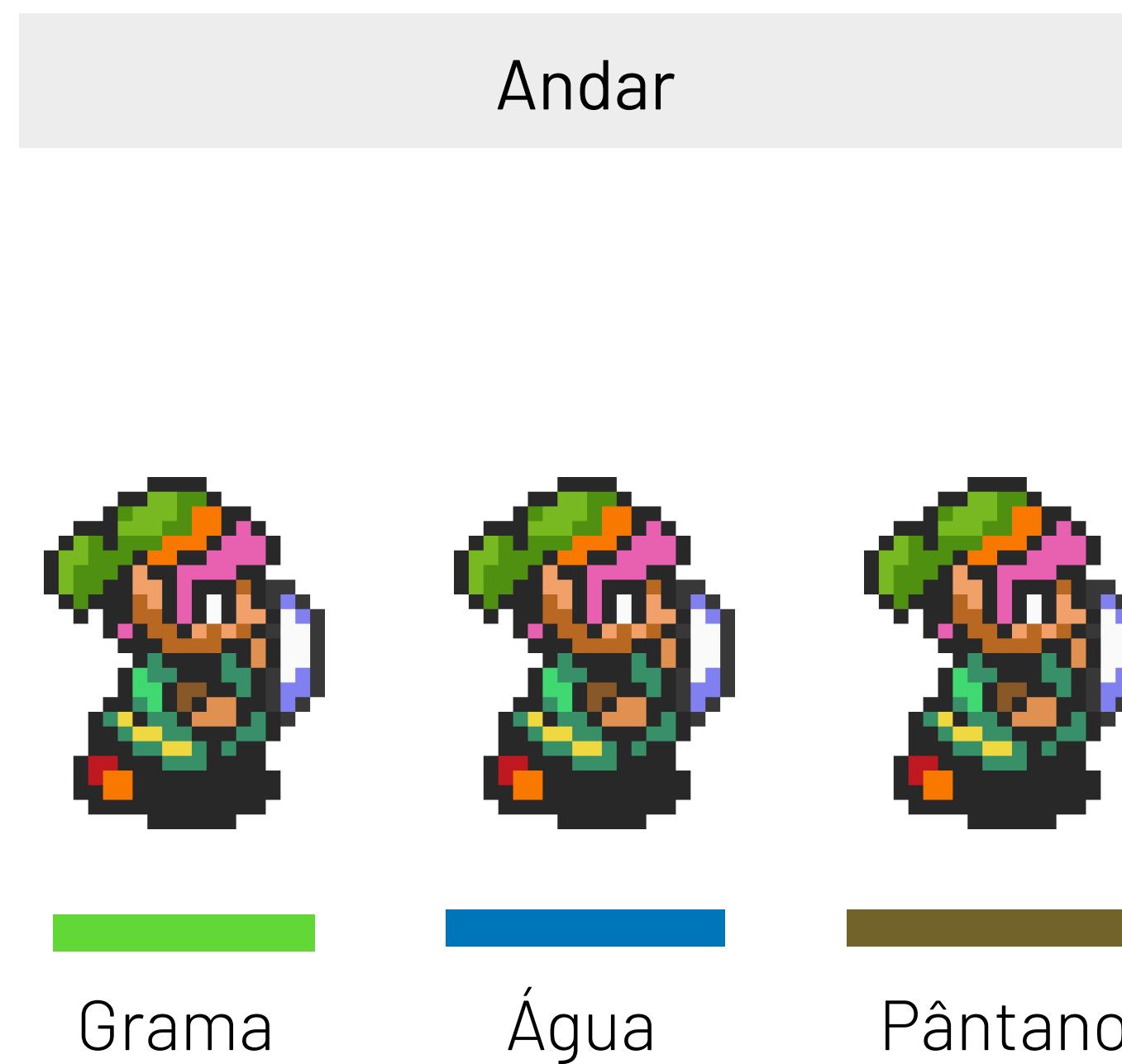
Para resolver esse problema, podemos associar mais de um som para a mesma animação:



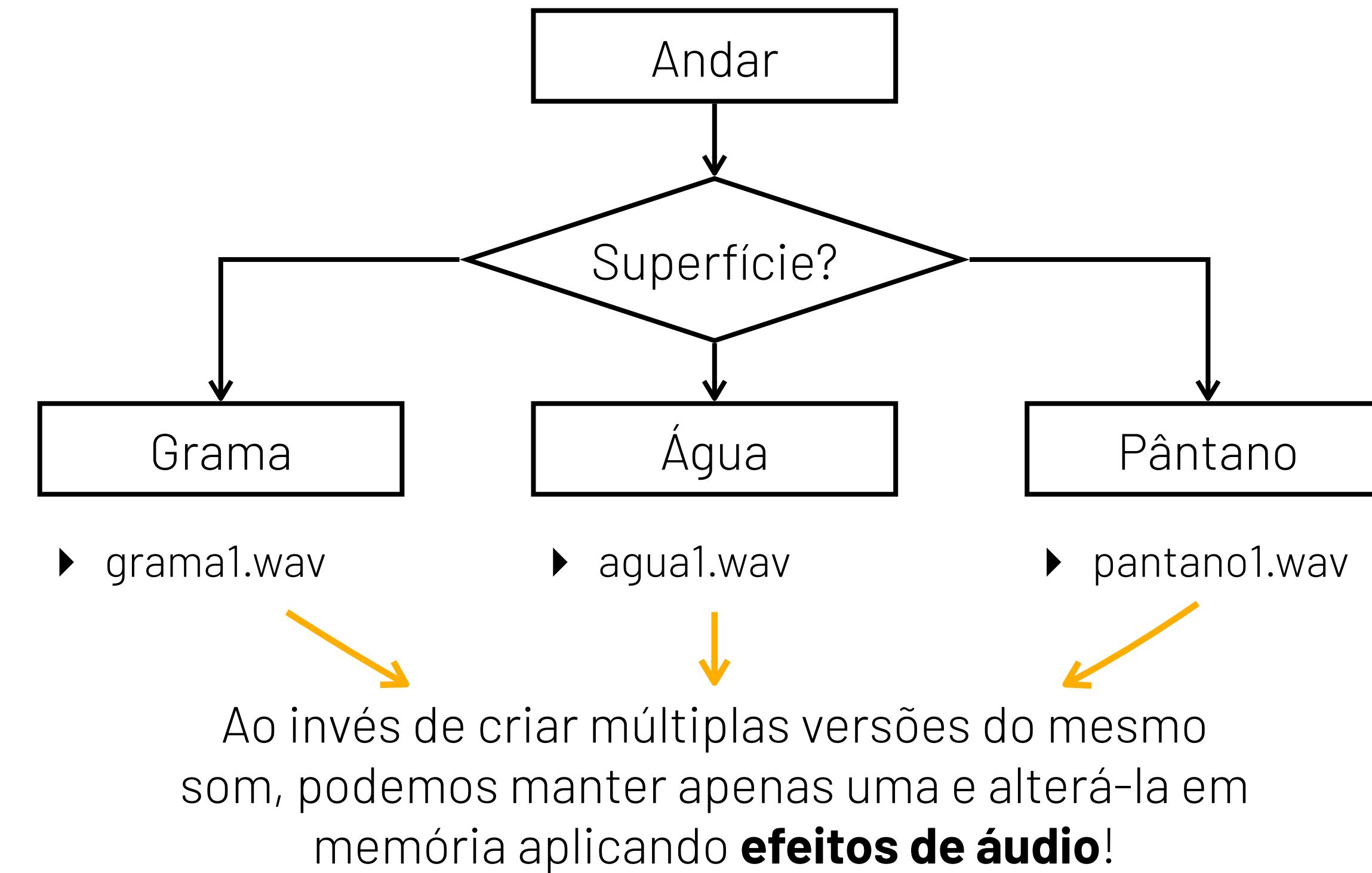
Lidando com Repetições



Quando reproduzimos o mesmo efeito de som repetidas vezes, como é o caso de animações recorrentes no jogo, ele se torna rapidamente enjoativo.



Para resolver esse problema, podemos associar mais de um som para a mesma animação:



Efeitos de Processamento de Áudio



Em processamento de sinais, um **efeito** é uma transformação feita no sinal de áudio para fins estéticos ou de melhoria de qualidade. Entre os efeitos muitos mais comuns em jogos estão:

Efeito	Definição	Exemplo de Uso
Reverb	Simula reflexões sonoras em um espaço (como um corredor ou sala)	Simular o eco de sons altos em lugares fechados
Filtro passa-baixo	Simular sons passando por paredes	Simular sons passando por paredes
Compressor	Normalizar os níveis de volume de arquivos de som diferentes	Normalizar os níveis de volume de arquivos de som diferentes
Pitch Shift (Deslocamento de Tom)	Aumenta ou diminui o tom sem alterar a velocidade	Adicionar variação sonora em uma animação

Próxima aula



A17: Composição Musical

- ▶ Efeitos sonoros
 - ▶ Problemas de Repetição
- ▶ Músicas de Fundo
- ▶ Lidando com múltiplos canais