

# Table of Contents

16 Games in C++ .....	2
Configuração do Ambiente .....	5
Requisitos do Sistema .....	8
Instalação do SFML .....	13
Instalação do CMake .....	21
Configuração da IDE .....	30
Configuração Final do Ambiente .....	40
Compilação .....	49
Troubleshooting .....	59

# 16 Games in C++

Bem-vindo à documentação do projeto **16 Games in C++!** 🎮

Este projeto é uma coleção de 16 jogos clássicos implementados em C++ utilizando a biblioteca SFML (Simple and Fast Multimedia Library). Cada jogo é completo e funcional, perfeito para aprender conceitos de programação de jogos e C++.

## Sobre o Projeto

O projeto **16 Games in C++** foi criado com o objetivo de demonstrar diferentes técnicas de programação de jogos, desde jogos simples como Tetris até jogos mais complexos como Chess e Asteroids. Todos os jogos utilizam:

- **C++17** como linguagem de programação
- **SFML 2.5+** para gráficos, áudio e entrada
- **CMake** para build system
- Estrutura modular e código limpo

## Jogos Incluídos

1. **Tetris** - O clássico jogo de blocos
2. **Doodle Jump** - Pule o mais alto possível
3. **Arkanoid** - Quebre todos os blocos
4. **Snake** - A serpente clássica
5. **Minesweeper** - Campo minado
6. **Fifteen Puzzle** - Quebra-cabeça numérico
7. **Racing (Top Down)** - Corrida vista de cima
8. **Outrun** - Corrida em perspectiva
9. **Xonix** - Conquiste território
10. **Bejeweled** - Combine joias
11. **NetWalk** - Conecte os canos
12. **Mahjong Solitaire** - Paciência mahjong

13. **Tron** - Batalha de luz
14. **Chess** - Xadrez completo
15. **Volleyball** - Vôlei arcade
16. **Asteroids** - Destrua os asteroides



## Início Rápido

Para começar rapidamente:

1. Configure o ambiente ([Configuração do Ambiente](#))
2. Clone o repositório
3. Execute o script de setup: `./setup.sh`
4. Compile e jogue: `make all_games`



## Estrutura da Documentação

Esta documentação está organizada nas seguintes seções:

- **Configuração do Ambiente** ([Configuração do Ambiente](#)) - Como instalar e configurar tudo que você precisa
- **Compilação** ([Compilação](#)) - Guias de build e execução
  - - Como o código está organizado
  - - Documentação específica de cada jogo
- **Troubleshooting** ([Troubleshooting](#)) - Soluções para problemas comuns



## Requisitos do Sistema

- **Sistema Operacional:** Linux, macOS ou Windows
- **Compilador:** GCC 7+ ou Clang 6+ com suporte a C++17
- **CMake:** 3.10 ou superior
- **SFML:** 2.5 ou superior



## Contribuindo

Este projeto é open source! Você pode:

- Reportar bugs
- Sugerir melhorias
- Contribuir com código
- Melhorar a documentação



## Licença

Este projeto é distribuído sob a licença MIT. Veja o arquivo LICENSE para mais detalhes.

**Pronto para começar?** Vá para Configuração do Ambiente ([Configuração do Ambiente](#)) e configure seu sistema para executar os jogos!

# Configuração do Ambiente

Este guia te ajudará a configurar completamente o ambiente de desenvolvimento para executar os 16 jogos em C++. 🛠️

## Visão Geral

Para executar este projeto, você precisará instalar e configurar:

1. **Requisitos do Sistema** ([Requisitos do Sistema](#)) - Verificar compatibilidade
2. **SFML** ([Instalação do SFML](#)) - Biblioteca gráfica principal
3. **CMake** ([Instalação do CMake](#)) - Sistema de build
4. **IDE** ([Configuração da IDE](#)) - Ambiente de desenvolvimento (opcional)
5. **Configuração Final** ([Configuração Final do Ambiente](#)) - Testes e validação







## Setup Automático (Recomendado)

O projeto inclui um script de configuração automática que verifica e configura tudo para você:

```
# Clone o repositório
git clone <repository-url>
cd 16Games-in-Cpp

# Execute o script de setup
chmod +x setup.sh
./setup.sh
```

O script `setup.sh` irá:

-  Verificar se o SFML está instalado
-  Verificar se o CMake está disponível
-  Verificar se há um compilador C++ válido
-  Configurar o projeto com CMake
-  Compilar um jogo de teste
-  Mostrar instruções de uso

## Setup Manual

Se preferir instalar manualmente ou se o script automático falhar, siga os guias específicos:

### 1. Requisitos do Sistema

Primeiro, verifique se seu sistema atende aos requisitos mínimos ([Requisitos do Sistema](#)).

### 2. Instalar SFML

A biblioteca SFML é essencial para gráficos, áudio e entrada. Veja o guia de instalação do SFML ([Instalação do SFML](#)).

### 3. Instalar CMake

O CMake é usado para gerenciar o build do projeto. Veja o guia de instalação do CMake ([Instalação do CMake](#)).

### 4. Configurar IDE (Opcional)

Para uma melhor experiência de desenvolvimento, configure sua IDE preferida ([Configuração da IDE](#)).

### 5. Configuração Final

Complete a configuração do ambiente ([Configuração Final do Ambiente](#)) e teste tudo.

## Verificação Rápida

Após a instalação, execute estes comandos para verificar se tudo está funcionando:

```
# Verificar SFML
pkg-config --exists sfml-all && echo "✅ SFML OK" || echo "❌ SFML não encontrado"

# Verificar CMake
cmake --version && echo "✅ CMake OK" || echo "❌ CMake não encontrado"

# Verificar compilador
g++ --version && echo "✅ G++ OK" || echo "❌ G++ não encontrado"
```

## Próximos Passos

Após configurar o ambiente:

1. Vá para Compilação ([Compilação](#)) para aprender a compilar os jogos
2. Ou vá direto para para executar rapidamente


## ! Problemas?

Se encontrar algum problema durante a configuração:

1. Consulte o Troubleshooting ([Troubleshooting](#))
2. Verifique se seguiu todos os passos corretamente
3. Confirme se seu sistema atende aos requisitos mínimos

**Dica:** O script `setup.sh` é a forma mais rápida e confiável de configurar o ambiente. Use-o sempre que possível!

# Requisitos do Sistema

Antes de instalar o projeto 16 Games in C++, verifique se seu sistema atende aos requisitos mínimos. 

## Sistemas Operacionais Suportados

### Linux (Recomendado)

- **Ubuntu:** 18.04 LTS ou superior
- **Debian:** 10 (Buster) ou superior
- **Fedora:** 30 ou superior
- **Arch Linux:** Versão atual
- **openSUSE:** Leap 15.2 ou superior
- **CentOS/RHEL:** 8 ou superior

### macOS

- **macOS:** 10.14 (Mojave) ou superior
- **Xcode:** 10 ou superior (para compilador)

### Windows

- **Windows:** 10 ou superior
- **Visual Studio:** 2019 ou superior
- **MinGW-w64:** Como alternativa ao Visual Studio

## Compilador C++

### Requisitos Mínimos

- **Suporte ao C++17:** Obrigatório
- **Versões mínimas:**
  - **GCC:** 7.0 ou superior
  - **Clang:** 6.0 ou superior



- **MSVC:** Visual Studio 2019 ou superior

## Verificação do Compilador

### Linux/macOS

```
# Verificar GCC
g++ --version
# Deve mostrar versão 7.0+

# Verificar Clang (se disponível)
clang++ --version
# Deve mostrar versão 6.0+
```

### Windows

```
# Visual Studio
cl
# Deve mostrar MSVC 19.20+

# MinGW
g++ --version
# Deve mostrar versão 7.0+
```

## Dependências Principais

### CMake

- **Versão mínima:** 3.10
- **Recomendada:** 3.16 ou superior

### SFML

- **Versão mínima:** 2.5.0
- **Recomendada:** 2.5.1 ou superior
- **Módulos necessários:**
  - sfml-system
  - sfml-window
  - sfml-graphics

- sfml-audio
- sfml-network (opcional)

## Bibliotecas do Sistema (Linux)

```
# Ubuntu/Debian
sudo apt-get install libsFML-dev cmake build-essential

# Fedora
sudo dnf install SFML-devel cmake gcc-c++

# Arch Linux
sudo pacman -S sfml cmake gcc
```



## Espaço em Disco

### Requisitos de Espaço

- **Código fonte:** ~50 MB
- **Dependências:** ~200 MB (SFML + CMake)
- **Build completo:** ~100 MB
- **Total recomendado:** ~500 MB livre

### Estrutura de Diretórios

```
16Games-in-Cpp/
├─ build/           # ~100 MB (arquivos compilados)
├─ Writerside/      # ~10 MB (documentação)
├─ games/           # ~40 MB (código fonte dos jogos)
└─ assets/          # ~5 MB (imagens, sons, fonts)
```



## Hardware Recomendado

### Mínimo

- **CPU:** Dual-core 2.0 GHz
- **RAM:** 4 GB
- **GPU:** Integrada com OpenGL 2.1

- **Resolução:** 1024x768

## Recomendado

- **CPU:** Quad-core 2.5 GHz ou superior
- **RAM:** 8 GB ou superior
- **GPU:** Dedicada com OpenGL 3.3+
- **Resolução:** 1920x1080 ou superior








## Verificação Automática

Use o script de verificação incluído no projeto:

```
# Executar verificação de requisitos
./setup.sh

# Ou verificar manualmente
./check_requirements.sh # Se disponível
```

O script verificará:

-  Sistema operacional compatível
-  Compilador C++ com suporte ao C++17
-  CMake versão adequada
-  SFML instalado e funcional
-  Espaço em disco suficiente



## Problemas Comuns

### Compilador Muito Antigo

```
# Ubuntu 18.04 - instalar GCC mais novo
sudo apt update
sudo apt install gcc-9 g++-9
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 90
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-9 90
```

### SFML Não Encontrado

```
# Verificar instalação
pkg-config --exists sfml-all
pkg-config --modversion sfml-all

# Se não encontrado, reinstalar
sudo apt-get install --reinstall libsFML-dev
```

## CMake Muito Antigo

```
# Ubuntu - instalar versão mais nova
wget https://github.com/Kitware/CMake/releases/download/v3.20.0/cmake-3.20.0-
Linux-x86_64.sh
chmod +x cmake-3.20.0-Linux-x86_64.sh
sudo ./cmake-3.20.0-Linux-x86_64.sh --prefix=/usr/local --skip-license
```

## Próximos Passos

Se seu sistema atende aos requisitos:

1. Prosiga para Instalação do SFML ([Instalação do SFML](#))
2. Ou execute o setup automático ([Configuração do Ambiente](#))

Se encontrou problemas:

1. Consulte o Troubleshooting ([Troubleshooting](#))
2. Atualize seu sistema e tente novamente

**Dica:** Na dúvida, execute `./setup.sh` - o script detectará automaticamente se há algum problema com os requisitos!

# Instalação do SFML

O SFML (Simple and Fast Multimedia Library) é a biblioteca principal usada por todos os 16 jogos. Este guia mostra como instalá-la em diferentes sistemas operacionais. 🎮



## O que é SFML?

SFML é uma biblioteca C++ que fornece:

- **Gráficos 2D** - Desenho de sprites, formas, texto
- **Áudio** - Reprodução de sons e música
- **Janelas** - Criação e gerenciamento de janelas
- **Eventos** - Captura de teclado, mouse e joystick
- **Rede** - Comunicação TCP/UDP (não usado neste projeto)



## Linux

### Ubuntu/Debian

```
# Atualizar repositórios
sudo apt update

# Instalar SFML e dependências de desenvolvimento
sudo apt install libsFML-dev

# Verificar instalação
pkg-config --modversion sfml-all
```

### Fedora/CentOS/RHEL

```
# Fedora
sudo dnf install SFML-devel

# CentOS/RHEL (com EPEL)
sudo yum install epel-release
sudo yum install SFML-devel

# Verificar instalação
pkg-config --modversion sfml-all
```

## Arch Linux

```
# Instalar SFML
sudo pacman -S sfml

# Verificar instalação
pkg-config --modversion sfml-all
```

## openSUSE

```
# openSUSE Leap/Tumbleweed
sudo zypper install libsFML2-devel

# Verificar instalação
pkg-config --modversion sfml-all
```

## macOS

### Usando Homebrew (Recomendado)

```
# Instalar Homebrew se não tiver
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Instalar SFML
brew install sfml

# Verificar instalação
pkg-config --modversion sfml-all
```

### Usando MacPorts

```
# Instalar SFML
sudo port install sfml

# Verificar instalação
pkg-config --modversion sfml-all
```

## Instalação Manual

1. Baixe SFML do site oficial (<https://www.sfml-dev.org/download.php>)
2. Extraia para `/usr/local/`

3. Configure as variáveis de ambiente:

```
export SFML_ROOT=/usr/local/SFML-2.5.1
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$SFML_ROOT/lib/pkgconfig
```

## Windows

### Visual Studio (Recomendado)

#### 1. Baixar SFML

- Acesse SFML Downloads (<https://www.sfml-dev.org/download.php>)
- Baixe a versão para Visual Studio (ex: SFML-2.5.1-windows-vc15-64-bit.zip)

#### 2. Extrair e Configurar

```
# Extrair para C:\SFML
# Estrutura deve ficar:
C:\SFML\
├─ bin\
├─ include\
├─ lib\
└─ examples\
```

#### 3. Configurar Projeto Visual Studio

No seu projeto CMake ou Visual Studio:

```
set(SFML_ROOT "C:/SFML")
find_package(SFML 2.5 COMPONENTS system window graphics audio REQUIRED)
```

### MinGW-w64

```
# Usando MSYS2
pacman -S mingw-w64-x86_64-sfml

# Verificar instalação
pkg-config --modversion sfml-all
```

### vcpkg (Alternativa)

```
# Instalar vcpkg
git clone https://github.com/Microsoft/vcpkg.git
```

```
cd vcpkg
.\bootstrap-vcpkg.bat

# Instalar SFML
.\vcpkg install sfml:x64-windows
```

## Compilação desde o Código Fonte

Se os pacotes pré-compilados não funcionarem, compile o SFML:

### Linux/macOS

```
# Baixar código fonte
git clone https://github.com/SFML/SFML.git
cd SFML

# Criar diretório de build
mkdir build
cd build

# Configurar com CMake
cmake .. -DCMAKE_INSTALL_PREFIX=/usr/local

# Compilar
make -j$(nproc)

# Instalar (pode precisar de sudo)
sudo make install

# Atualizar cache de bibliotecas (Linux)
sudo ldconfig
```

### Windows (Visual Studio)

```
# No Developer Command Prompt
git clone https://github.com/SFML/SFML.git
cd SFML
mkdir build
cd build

cmake .. -G "Visual Studio 16 2019" -A x64
cmake --build . --config Release
cmake --install . --prefix C:\SFML
```



## ✓ Verificação da Instalação

### Teste Básico

Crie um arquivo `test_sfml.cpp`:

```
#include <SFML/Graphics.hpp>
#include <iostream>

int main() {
    // Tentar criar uma janela
    sf::RenderWindow window(sf::VideoMode(200, 200), "SFML Test");

    if (!window.isOpen()) {
        std::cout << "✗ Erro ao criar janela SFML\n";
        return 1;
    }

    std::cout << "✓ SFML funcionando corretamente!\n";

    // Fechar imediatamente
    window.close();
    return 0;
}
```

### Compilar Teste

```
# Linux/macOS
g++ -o test_sfml test_sfml.cpp $(pkg-config --cflags --libs sfml-all)

# Windows (MinGW)
g++ -o test_sfml.exe test_sfml.cpp -lsfml-graphics -lsfml-window -lsfml-system

# Executar
./test_sfml
```

### Usando CMake (Recomendado)

Crie um `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 3.10)
project(TestSFML)

set(CMAKE_CXX_STANDARD 17)
```

```
find_package(PkgConfig REQUIRED)
pkg_check_modules(SFML REQUIRED sfml-all>=2.5)

add_executable(test_sfml test_sfml.cpp)
target_link_libraries(test_sfml ${SFML_LIBRARIES})
target_compile_options(test_sfml PRIVATE ${SFML_CFLAGS})
```

```
mkdir build
cd build
cmake ..
make
./test_sfml
```

## Verificação com pkg-config

```
# Verificar se SFML está disponível
pkg-config --exists sfml-all
echo $? # Deve retornar 0

# Ver versão instalada
pkg-config --modversion sfml-all

# Ver flags de compilação
pkg-config --cflags sfml-all

# Ver flags de link
pkg-config --libs sfml-all

# Informações completas
pkg-config --cflags --libs sfml-all
```

## Solução de Problemas

### Erro: "sfml-all not found"

```
# Verificar se pkg-config está instalado
pkg-config --version

# Listar todos os pacotes disponíveis
pkg-config --list-all | grep -i sfml
```

```
# Verificar caminhos do pkg-config
echo $PKG_CONFIG_PATH

# Linux: SFML pode estar em /usr/lib/pkgconfig/
ls /usr/lib/pkgconfig/ | grep sfml
```

## Erro de Linking

```
# Verificar se bibliotecas estão instaladas
ldconfig -p | grep sfml # Linux
find /usr -name "*sfml*" 2>/dev/null # Linux/macOS

# Windows: verificar PATH
echo %PATH%
```

## Versão Incompatível

```
# Desinstalar versão antiga
sudo apt remove libsFML-dev # Ubuntu
brew uninstall sfml         # macOS

# Limpar cache
sudo apt autoremove
brew cleanup

# Reinstalar versão correta
sudo apt install libsFML-dev
brew install sfml
```

## Próximos Passos

Após instalar o SFML com sucesso:

1. Prosiga para Instalação do CMake ([Instalação do CMake](#))
2. Ou volte para Configuração do Ambiente ([Configuração do Ambiente](#))

Se ainda tiver problemas:

1. Consulte o Troubleshooting ([Troubleshooting](#))
2. Execute o script automático: `./setup.sh`

**Dica:** O comando `pkg-config --cflags --libs sfml-all` mostra exatamente como compilar com SFML. Guarde essa informação!

# Instalação do CMake

O CMake é o sistema de build usado pelo projeto 16 Games in C++. Este guia mostra como instalá-lo em diferentes sistemas operacionais. 🛠️



## O que é CMake?

CMake é uma ferramenta multiplataforma que:

- **Gera arquivos de build** para diferentes sistemas (Make, Visual Studio, Xcode)
- **Gerencia dependências** de forma automática
- **Configura compilação** com diferentes opções
- **Simplifica builds** em múltiplas plataformas



## Versões Suportadas

- **Mínima:** 3.10
- **Recomendada:** 3.16 ou superior
- **Ideal:** 3.20 ou superior (melhor suporte ao C++17)



## Linux

### Ubuntu/Debian

#### Versão dos Repositórios (Mais Simples)

```
# Atualizar repositórios
sudo apt update

# Instalar CMake
sudo apt install cmake

# Verificar versão
cmake --version
```

#### Versão Mais Recente (Recomendado)

```
# Adicionar repositório Kitware (oficial)
wget -O - https://apt.kitware.com/keys/kitware-archive-latest.asc 2>/dev/null
| gpg --dearmor - | sudo tee /etc/apt/trusted.gpg.d/kitware.gpg >/dev/null
```

```
# Ubuntu 20.04
sudo apt-add-repository 'deb https://apt.kitware.com/ubuntu/ focal main'

# Ubuntu 22.04
sudo apt-add-repository 'deb https://apt.kitware.com/ubuntu/ jammy main'

# Instalar
sudo apt update
sudo apt install cmake

# Verificar versão
cmake --version
```

## Fedora/CentOS/RHEL

```
# Fedora
sudo dnf install cmake

# CentOS/RHEL 8+
sudo dnf install cmake

# Verificar versão
cmake --version
```

## Arch Linux

```
# Instalar CMake
sudo pacman -S cmake

# Verificar versão
cmake --version
```

## openSUSE

```
# openSUSE Leap/Tumbleweed
sudo zypper install cmake

# Verificar versão
cmake --version
```



## Usando Homebrew (Recomendado)

```
# Instalar Homebrew se não tiver
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Instalar CMake
brew install cmake

# Verificar versão
cmake --version
```

## Usando MacPorts

```
# Instalar CMake
sudo port install cmake

# Verificar versão
cmake --version
```

## Instalação Manual

```
# Baixar binário
curl -L -O https://github.com/Kitware/CMake/releases/download/v3.25.1/cmake-
3.25.1-macos-universal.tar.gz

# Extrair
tar -xzf cmake-3.25.1-macos-universal.tar.gz

# Mover para aplicações
sudo mv cmake-3.25.1-macos-universal/CMake.app /Applications/

# Criar link simbólico
sudo ln -s /Applications/CMake.app/Contents/bin/cmake /usr/local/bin/cmake

# Verificar versão
cmake --version
```

## Windows

### Usando winget (Windows 10/11)

```
# Instalar CMake
winget install Kitware.CMake

# Verificar versão (reiniciar terminal)
cmake --version
```

## Usando Chocolatey

```
# Instalar Chocolatey se não tiver
# Ver: https://chocolatey.org/install

# Instalar CMake
choco install cmake

# Verificar versão
cmake --version
```

## Instalação Manual

1. Acesse CMake Downloads (<https://cmake.org/download/>)
2. Baixe o instalador Windows (.msi)
3. Execute o instalador
4. **Importante:** Marque "Add CMake to system PATH"
5. Verificar instalação:

```
cmake --version
```

## Visual Studio

O Visual Studio 2019/2022 já inclui CMake:

- Está em C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE\CommonExtensions\Microsoft\CMake\CMake\bin\cmake.exe
- Pode ser necessário adicionar ao PATH manualmente



## Compilação desde o Código Fonte

Se precisar de uma versão específica ou os pacotes não funcionarem:

## Linux/macOS



```
# Baixar código fonte
wget https://github.com/Kitware/CMake/releases/download/v3.25.1/cmake-3.25.1.tar.gz
tar -xzf cmake-3.25.1.tar.gz
cd cmake-3.25.1

# Configurar (bootstrap)
./bootstrap --prefix=/usr/local

# Compilar
make -j$(nproc)

# Instalar
sudo make install

# Verificar versão
cmake --version
```

### Compilação Rápida (Sem Bootstrap)

```
# Se já tiver CMake instalado (versão mais antiga)
mkdir build
cd build
cmake ..
make -j$(nproc)
sudo make install
```

## Verificação da Instalação

### Teste Básico

```
# Verificar versão
cmake --version

# Deve mostrar algo como:
# cmake version 3.25.1
```

### Teste de Funcionalidade

Crie um projeto de teste:

**CMakeLists.txt**

```
cmake_minimum_required(VERSION 3.10)
project(TestCMake)

set(CMAKE_CXX_STANDARD 17)

add_executable(test_cmake main.cpp)
```

### main.cpp

```
#include <iostream>
int main() {
    std::cout << "✅ CMake funcionando!\n";
    return 0;
}
```

### Compilar e Testar

```
mkdir build
cd build
cmake ..
make # ou cmake --build .
./test_cmake
```

## Configurações Úteis

### Configurar CMake Globalmente

```
# Criar arquivo de configuração
mkdir -p ~/.cmake
cat > ~/.cmake/CMakeCache.txt << EOF
CMAKE_BUILD_TYPE:STRING=Release
CMAKE_CXX_STANDARD:STRING=17
CMAKE_EXPORT_COMPILE_COMMANDS:BOOL=ON
EOF
```

### Aliases Úteis

Adicione ao seu `.bashrc` ou `.zshrc`:

```
# Aliases para CMake
alias cb='cmake --build .'
alias cc='cmake ..'
```

```
alias cr='cmake .. && make'
alias ctest='ctest --output-on-failure'
```

## Ferramentas Complementares

### CMake GUI (Opcional)

```
# Linux
sudo apt install cmake-qt-gui # Ubuntu/Debian
sudo dnf install cmake-gui    # Fedora

# macOS
brew install --cask cmake

# Windows - incluído no instalador
```

### ccmake (Curses Interface)

```
# Linux - geralmente incluído
ccmake ..

# Navegar com setas, Enter para editar, 'c' para configurar, 'g' para gerar
```

## Verificação de Recursos

```
# Ver todas as opções do CMake
cmake --help

# Ver geradores disponíveis
cmake --help | grep "Generators"

# Informações do sistema
cmake --system-information

# Verificar variáveis disponíveis
cmake --help-variable-list | head -20
```

## Solução de Problemas

### CMake Não Encontrado

```
# Verificar PATH
echo $PATH

# Linux/macOS - onde está o CMake?
which cmake
whereis cmake

# Windows
where cmake
```

## Versão Muito Antiga

```
# Desinstalar versão antiga
sudo apt remove cmake      # Ubuntu
brew uninstall cmake       # macOS

# Instalar versão mais recente (ver seções acima)
```

## Erro de Permissão

```
# Linux - problemas de permissão
sudo chown -R $USER:$USER ~/.cmake
sudo chmod -R 755 ~/.cmake
```

## Conflito de Versões

```
# Ver todas as versões instaladas
ls /usr/bin/cmake*
ls /usr/local/bin/cmake*

# Usar versão específica
/usr/local/bin/cmake --version
```

## CMake no Projeto 16 Games

No nosso projeto, o CMake:

- **Detecta SFML** automaticamente
- **Configura compilação** para C++17
- **Gerencia assets** (copia imagens, sons, etc.)

- Cria **targets** para cada jogo
- Oferece comandos de build e execução

## Comandos Principais

```
# Configurar projeto
cmake ..

# Compilar todos os jogos
make all_games

# Compilar jogo específico
make tetris

# Executar jogo
make run_tetris
```

## Próximos Passos

Após instalar o CMake com sucesso:

1. Prossiga para Configuração da IDE ([Configuração da IDE](#))
2. Ou pule para Configuração Final ([Configuração Final do Ambiente](#))
3. Ou volte para Configuração do Ambiente ([Configuração do Ambiente](#))

Se tiver problemas:

1. Consulte o Troubleshooting ([Troubleshooting](#))
2. Execute o script automático: `./setup.sh`


**Dica:** Use sempre `cmake --build .` em vez de `make` para máxima compatibilidade entre plataformas!

# Configuração da IDE


Embora você possa compilar e executar os jogos apenas com terminal, usar uma IDE melhora significativamente a experiência de desenvolvimento. Este guia mostra como configurar diferentes IDEs para o projeto. 🖥️

## 🎯 IDEs Recomendadas



### Para Iniciantes

- Visual Studio Code (" [Visual Studio Code](#)" in "[Configuração da IDE](#)") - Leve, extensível, gratuito
- Code::Blocks ([Configuração da IDE](#)) - Simples, focado em C++

### Para Desenvolvedores Experientes

- CLion ("[CLion](#)" in "[Configuração da IDE](#)") - Profissional, JetBrains
- Visual Studio ("[Visual Studio](#)" in "[Configuração da IDE](#)") - Windows, Microsoft
- Qt Creator (" [Qt Creator](#)" in "[Configuração da IDE](#)") - Multiplataforma, excelente CMake

### Editores Avançados

- Vim/Neovim (" [Vim/Neovim](#)" in "[Configuração da IDE](#)") - Para usuários experientes
- Emacs (" [Emacs](#)" in "[Configuração da IDE](#)") - Altamente customizável

## Visual Studio Code

### Instalação

```
# Linux (Ubuntu/Debian)
sudo apt update
sudo apt install code

# macOS
brew install --cask visual-studio-code

# Windows - baixar do site oficial
# https://code.visualstudio.com/
```

### Extensões Essenciais

Instale essas extensões (Ctrl+Shift+X):

1. **C/C++** (Microsoft) - Suporte básico ao C++
2. **CMake Tools** (Microsoft) - Integração com CMake
3. **CMake** (twxs) - Syntax highlighting para CMake
4. **C++ Intellisense** (austin) - Melhor autocomplete
5. **GitLens** (GitKraken) - Integração Git avançada

## Configuração do Projeto

### 1. Abrir Projeto

```
cd 16Games-in-Cpp  
code .
```

### 2. Configurar CMake Tools

Pressione **Ctrl+Shift+P** e digite "CMake: Configure":

```
// .vscode/settings.json  
{  
  "cmake.configureSettings": {  
    "CMAKE_BUILD_TYPE": "Debug",  
    "CMAKE_CXX_STANDARD": "17"  
  },  
  "cmake.buildDirectory": "${workspaceFolder}/build",  
  "cmake.generator": "Unix Makefiles",  
  "C_Cpp.default.configurationProvider": "ms-vscode.cmake-tools"  
}
```

### 3. Configurar Tarefas de Build

```
// .vscode/tasks.json  
{  
  "version": "2.0.0",  
  "tasks": [  
    {  
      "label": "CMake Configure",  
      "type": "shell",  
      "command": "cmake",  
      "args": [".", "-DCMAKE_BUILD_TYPE=Debug"],  
      "options": {  
        "cwd": "${workspaceFolder}/build"  
      }  
    }  
  ]  
}
```

```

    },
    "group": "build"
  },
  {
    "label": "Build All Games",
    "type": "shell",
    "command": "make",
    "args": ["all_games"],
    "options": {
      "cwd": "${workspaceFolder}/build"
    },
    "group": {
      "kind": "build",
      "isDefault": true
    }
  }
]
}

```

#### 4. Configurar Launch (Debug)

```

// .vscode/launch.json
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Debug Tetris",
      "type": "cppdbg",
      "request": "launch",
      "program": "${workspaceFolder}/build/games/tetris/tetris",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${workspaceFolder}/build/games/tetris",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "setupCommands": [
        {
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        }
      ]
    }
  ]
}

```



```
]
}
```



## Configuração

1. **Abrir Projeto:** File → Open → Selecionar pasta do projeto
2. **CMake será detectado automaticamente**
3. **Configurar Build Types:**
  - File → Settings → Build, Execution, Deployment → CMake
  - Adicionar Debug e Release profiles

## Configurações Recomendadas

### CMake Settings

```
Debug Profile:
- Build type: Debug
- CMake options: -DCMAKE_BUILD_TYPE=Debug
- Build directory: build/Debug

Release Profile:
- Build type: Release
- CMake options: -DCMAKE_BUILD_TYPE=Release
- Build directory: build/Release
```

### Run Configurations

CLion criará automaticamente configurações para cada jogo. Você pode:

- **Executar:** Ctrl+Shift+F10
- **Debug:** Shift+F9
- **Configurar:** Run → Edit Configurations

## Plugins Úteis

- **Valgrind Memcheck** - Detecção de vazamentos
- **CPU Usage Indicator** - Monitor de performance

- Rainbow Brackets – Melhor visualização

## Visual Studio (Windows)

### Configuração

1. **Instalar:** Visual Studio Community (gratuito)
2. **Workloads:** Marcar "Desktop development with C++"
3. **Componentes:** Incluir CMake tools

### Abrir Projeto

1. File → Open → CMake...
2. Selecionar `CMakeLists.txt` na raiz do projeto
3. Visual Studio configurará automaticamente

### Configurações CMake

```
// CMakeSettings.json (criado automaticamente)
{
  "configurations": [
    {
      "name": "x64-Debug",
      "generator": "Ninja",
      "configurationType": "Debug",
      "buildRoot": "${projectDir}\\build\\${name}",
      "installRoot": "${projectDir}\\install\\${name}",
      "cmakeCommandArgs": "",
      "buildCommandArgs": "",
      "ctestCommandArgs": ""
    }
  ]
}
```

## Qt Creator

### Instalação

```
# Linux
sudo apt install qtcreator
```

```
# macOS
brew install --cask qt-creator

# Windows - baixar do site Qt
```

## Configuração

1. **Abrir:** File → Open File or Project → CMakeLists.txt
2. **Kit Selection:** Escolher kit apropriado (GCC/Clang)
3. **Build Directory:** Configurar diretório de build

## Vantagens

- Excelente suporte CMake
- Debugger integrado
- Profiler built-in
- Git integration



## Code::Blocks

### Instalação

```
# Linux
sudo apt install codeblocks

# Windows/macOS - baixar do site oficial
```

## Importar Projeto CMake

1. File → Import → Import CMake Project
2. Selecionar CMakeLists.txt
3. Configurar compilador e opções



## Vim/Neovim

### Plugins Recomendados (com vim-plug)

```

" .vimrc ou init.vim
call plug#begin()

" LSP e Completions
Plug 'neoclide/coc.nvim', {'branch': 'release'}
Plug 'clangd/coc-clangd'

" CMake
Plug 'cdelledonne/vim-cmake'
Plug 'vhdirk/vim-cmake'

" Syntax
Plug 'vim-syntastic/syntastic'
Plug 'octol/vim-cpp-enhanced-highlight'

" Git
Plug 'tpope/vim-fugitive'

call plug#end()

```

## Configuração CMake

```

" Keybindings para CMake
nnoremap <leader>cc :CMake<CR>
nnoremap <leader>cb :CMakeBuild<CR>
nnoremap <leader>cr :CMakeRun<CR>

```

## Emacs

### Configuração com use-package

```

;; init.el
(use-package cmake-mode
  :ensure t)

(use-package cmake-ide
  :ensure t
  :config
  (cmake-ide-setup))

(use-package company
  :ensure t
  :config

```

```
(global-company-mode))

(use-package lsp-mode
  :ensure t
  :init
  (setq lsp-keymap-prefix "C-c l")
  :hook ((c++-mode . lsp))
  :commands lsp)
```

## Configuração Geral para Todas IDEs

### Formatação de Código

Crie `.clang-format` na raiz do projeto:

```
BasedOnStyle: Google
IndentWidth: 4
TabWidth: 4
UseTab: Never
ColumnLimit: 100
AllowShortFunctionsOnASingleLine: Empty
```

### EditorConfig

Crie `.editorconfig`:

```
root = true

[*]
charset = utf-8
end_of_line = lf
insert_final_newline = true
trim_trailing_whitespace = true

[*.{cpp,hpp,h}]
indent_style = space
indent_size = 4

[CMakeLists.txt,*.cmake]
indent_style = space
indent_size = 2
```

### Git Ignore para IDEs

Adicione ao `.gitignore`:

```
# IDEs
.vscode/
.idea/
*.user
*.pro.user*
.cproject
.project
.settings/
*.cbp
```

## Comandos Úteis por IDE

### VS Code

- Build: `Ctrl+Shift+P` → "CMake: Build"
- Run: `Ctrl+F5`
- Debug: `F5`
- Terminal: `Ctrl+`` (backtick)

### CLion

- Build: `Ctrl+F9`
- Run: `Shift+F10`
- Debug: `Shift+F9`
- Terminal: `Alt+F12`

### Visual Studio

- Build: `Ctrl+Shift+B`
- Run: `Ctrl+F5`
- Debug: `F5`
- Terminal: View → Terminal

## Dicas Gerais

### Performance

- Use SSD para diretório de build
- Configure RAM adequada (mínimo 8GB)
- Feche programas desnecessários durante compilação

## Produtividade

- Configure atalhos personalizados
- Use templates para novos arquivos
- Configure snippets para código comum
- Use git integration

## Debug

- Configure breakpoints nos pontos críticos
- Use watch variables para monitorar estado
- Ative otimização apenas em release



## Próximos Passos

Após configurar sua IDE:

1. Prossiga para Configuração Final ([Configuração Final do Ambiente](#))
2. Ou teste com
3. Volte para Configuração do Ambiente ([Configuração do Ambiente](#))

Se tiver problemas:

1. Consulte o Troubleshooting ([Troubleshooting](#))
2. Verifique se SFML e CMake estão funcionando

**Dica:** Se é iniciante, comece com VS Code. Se quer máxima produtividade, use CLion. Para projetos simples, Code::Blocks é suficiente!

# Configuração Final do Ambiente

Este é o último passo da configuração do ambiente. Aqui vamos integrar tudo o que foi instalado e fazer os testes finais. 🚩

## Checklist Pré-Requisitos

Antes de continuar, confirme que você já tem:

- ☒ Sistema compatível ([Requisitos do Sistema](#))
- ☒ SFML instalado ([Instalação do SFML](#))
- ☒ CMake instalado ([Instalação do CMake](#))
- ☒ IDE configurada ([Configuração da IDE](#)) (opcional)

## Setup Automático (Recomendado)

O método mais rápido e confiável:

```
# Clonar o repositório
git clone <repository-url>
cd 16Games-in-Cpp

# Executar script de configuração
chmod +x setup.sh
./setup.sh
```

O script fará:

1. ☒ Verificação de dependências
2. ☒ Configuração do CMake
3. ☒ Compilação de teste
4. ☒ Validação do ambiente

Se o script executar sem erros, **seu ambiente está pronto!** 🎉

## Setup Manual

Se preferir fazer manualmente ou se o script automático falhar:



## 1. Clonar e Preparar Projeto

```
# Clonar repositório
git clone <repository-url>
cd 16Games-in-Cpp

# Criar diretório de build
mkdir -p build
cd build
```

## 2. Configurar CMake

```
# Configurar projeto
cmake ..

# Verificar se não houve erros
echo $? # Deve retornar 0
```

Saída esperada:

```
-- The CXX compiler identification is GNU 9.4.0
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Found PkgConfig: /usr/bin/pkg-config (found version "0.29.2")
-- Checking for modules 'sfml-all>=2.5'
--   Found sfml-all, version 2.5.1
-- Configuring done
-- Generating done
-- Build files have been written to: /path/to/16Games-in-Cpp/build
```

## 3. Teste de Compilação

```
# Compilar um jogo de teste
make tetris

# Verificar se o executável foi criado
ls games/tetris/
# Deve mostrar: tetris (e possivelmente images/)
```

## 4. Teste de Execução

```
# Executar o jogo
cd games/tetris
```

```
./tetris

# Ou usar o target cmake
cd ../../ # voltar para build/
make run_tetris
```

Se o jogo abrir uma janela e funcionar, **tudo está perfeito!** 🎮

## ✅ Validação Completa

### Teste Todos os Componentes

#### 1. Verificar Todas as Dependências

```
# Script de verificação completa
cat > check_all.sh << 'EOF'
#!/bin/bash

echo "🔍 Verificação Completa do Ambiente"
echo "===== "

# Verificar compilador
if g++ --version &>/dev/null; then
    echo "✅ G++ - $(g++ --version | head -n1)"
else
    echo "❌ G++ não encontrado"
    exit 1
fi

# Verificar CMake
if cmake --version &>/dev/null; then
    echo "✅ CMake - $(cmake --version | head -n1)"
else
    echo "❌ CMake não encontrado"
    exit 1
fi

# Verificar SFML
if pkg-config --exists sfml-all; then
    echo "✅ SFML - $(pkg-config --modversion sfml-all)"
else
    echo "❌ SFML não encontrado"
    exit 1
fi
```

```
# Verificar espaço em disco
SPACE=$(df -BG . | tail -1 | awk '{print $4}' | sed 's/G//')
if [ $SPACE -gt 1 ]; then
    echo "✅ Espaço em disco - ${SPACE}GB disponível"
else
    echo "⚠️ Pouco espaço em disco - ${SPACE}GB disponível"
fi

echo ""
echo "🎉 Todos os componentes verificados com sucesso!"
EOF

chmod +x check_all.sh
./check_all.sh
```

## 2. Compilar Todos os Jogos

```
# No diretório build/
make all_games

# Verificar se todos foram compilados
ls games/
# Deve mostrar todos os 16 diretórios de jogos
```

## 3. Teste Rápido de Múltiplos Jogos

```
# Script para testar vários jogos
cat > test_games.sh << 'EOF'
#!/bin/bash

games=("tetris" "snake" "arkanoid" "doodle_jump")

for game in "${games[@]"; do
    echo "🎮 Testando $game..."
    cd "games/$game"
    timeout 3s "./$game" &>/dev/null
    if [ $? -eq 124 ]; then # timeout (esperado)
        echo "✅ $game - OK"
    else
        echo "❌ $game - ERRO"
    fi
    cd "../.."
done
```

EOF

```
chmod +x test_games.sh
./test_games.sh
```

## Configurações Opcionais

### Variáveis de Ambiente Úteis

Adicione ao seu `.bashrc` ou `.zshrc`:

```
# Alias para o projeto 16 Games
alias games-build='cd ~/16Games-in-Cpp/build && make all_games'
alias games-clean='cd ~/16Games-in-Cpp && rm -rf build && mkdir build'
alias games-run='cd ~/16Games-in-Cpp/build'

# Variáveis para desenvolvimento
export GAMES_PROJECT_ROOT="$HOME/16Games-in-Cpp"
export GAMES_BUILD_DIR="$GAMES_PROJECT_ROOT/build"

# Função para executar jogos rapidamente
play_game() {
    if [ -z "$1" ]; then
        echo "Uso: play_game <nome_do_jogo>"
        echo "Jogos disponíveis: tetris, snake, arkanoid, etc."
        return 1
    fi

    cd "$GAMES_BUILD_DIR/games/$1" && ".$1"
}
```

### Configuração de Performance

```
# Para compilação mais rápida
export CMAKE_BUILD_PARALLEL_LEVEL=$(nproc)
export MAKEFLAGS="-j$(nproc)"

# Para debug mais detalhado
export CMAKE_VERBOSE_MAKEFILE=ON
```

## Testando Todos os Jogos

### Script de Teste Completo

```

# Criar script de teste abrangente
cat > full_test.sh << 'EOF'
#!/bin/bash

cd "$(dirname "$0")/build"

echo "🎮 Teste Completo dos 16 Games in C++"
echo "===== "

games=(
    "tetris" "doodle_jump" "arkanoid" "snake" "minesweeper"
    "fifteen_puzzle" "racing" "outrun" "xonix" "bejeweled"
    "netwalk" "mahjong" "tron" "chess" "volleyball" "asteroids"
)

success=0
total=${#games[@]}

for game in "${games[@]"; do
    echo -n "Testando $game... "

    if [ -f "games/$game/$game" ]; then
        echo "✅ Compilado"
        ((success++))
    else
        echo "❌ Não encontrado"
    fi
done

echo ""
echo "📊 Resultado: $success/$total jogos compilados com sucesso"

if [ $success -eq $total ]; then
    echo "🎉 Todos os jogos estão funcionando perfeitamente!"
    echo ""
    echo "📌 Comandos para jogar:"
    echo "    cd build/games/tetris && ./tetris"
    echo "    make run_tetris"
    echo "    # ... e assim por diante"
else
    echo "⚠️ Alguns jogos não foram compilados. Execute 'make all_games' novamente."
fi

```

EOF

```
chmod +x full_test.sh
./full_test.sh
```

## Estrutura Final Esperada

Após a configuração completa, sua estrutura deve estar assim:

```
16Games-in-Cpp/
├── build/                # Arquivos compilados
│   ├── games/           # Executáveis dos jogos
│   │   ├── tetris/
│   │   │   ├── tetris    # Executável
│   │   │   └── images/   # Assets copiados
│   │   ├── snake/
│   │   └── ... (16 jogos)
│   └── CMakeCache.txt    # Cache do CMake
├── 01 Tetris/            # Código fonte
├── 02 Doodle Jump/
├── ... (código dos jogos)
├── CMakeLists.txt        # Configuração CMake
├── setup.sh              # Script de configuração
└── README.md
```

## Comandos de Manutenção

### Limpeza e Reconstrução

```
# Limpar build completo
rm -rf build
mkdir build
cd build
cmake ..
make all_games

# Ou usando script
../setup.sh
```

### Atualizar Dependências

```
# Ubuntu/Debian
sudo apt update && sudo apt upgrade libsFML-dev cmake
```

```
# Fedora
sudo dnf update SFML-devel cmake

# macOS
brew update && brew upgrade sfml cmake
```

## Verificação de Integridade

```
# Verificar arquivos corrompidos
find . -name "*.cpp" -exec g++ -fsyntax-only {} \;

# Verificar links das bibliotecas
ldd build/games/tetris/tetris # Linux
otool -L build/games/tetris/tetris # macOS
```

## ! Troubleshooting Final

### Problema: CMake não encontra SFML

```
# Verificar onde SFML está instalado
find /usr -name "*sfml*" 2>/dev/null

# Definir manualmente se necessário
cmake .. -DSFML_ROOT=/usr/local
```

### Problema: Jogos não executam

```
# Verificar dependências
ldd games/tetris/tetris

# Verificar se assets foram copiados
ls games/tetris/images/

# Executar com debug
gdb games/tetris/tetris
```

### Problema: Performance ruim

```
# Compilar em modo Release
cmake .. -DCMAKE_BUILD_TYPE=Release
make all_games
```





## Próximos Passos

Ambiente configurado com sucesso! Agora você pode:

1. **Jogar:** Execute `make run_tetris` para testar
2. **Compilar:** Vá para
3. **Desenvolver:** Explore a
4. **Documentar:** Veja os individuais

## Parabéns!

Se chegou até aqui com sucesso, você tem:

-  Ambiente completamente configurado
-  Todos os 16 jogos compilados
-  Ferramentas de desenvolvimento prontas
-  Scripts de manutenção configurados

Divirta-se jogando e explorando o código! 🎮

**Dica:** Mantenha os scripts `setup.sh` e `full_test.sh` à mão. Eles serão úteis para manutenção futura do ambiente!



# Compilação

Este guia completo mostra como compilar, executar e gerenciar o build dos 16 jogos em C++. 🛠️

## Visão Geral

O projeto usa **CMake** como sistema de build, que oferece:

- **Detecção automática** de dependências
- **Build multiplataforma** (Linux, macOS, Windows)
- **Gerenciamento de assets** (imagens, sons, fonts)
- **Targets individuais** para cada jogo
- **Comandos simplificados** de execução

## Compilação Rápida

### Primeiro Build

```
# Configurar e compilar tudo
./setup.sh

# Ou manualmente:
mkdir build && cd build
cmake ..
make all_games
```

### Builds Subsequentes

```
cd build
make all_games
```

## Compilação por Jogo

### Jogos Disponíveis

```
# Listar todos os targets disponíveis
make help | grep -E "(tetris|snake|arkanoid)"

# Targets dos jogos:
tetris, doodle_jump, arkanoid, snake, minesweeper
```

```
fifteen_puzzle, racing, outrun, xonix, bejeweled  
netwalk, mahjong, tron, chess, volleyball, asteroids
```

## Compilar Jogo Específico

```
# Compilar apenas o Tetris  
make tetris  
  
# Compilar Snake  
make snake  
  
# Compilar Arkanoid  
make arkanoid
```

## Executar Jogos

```
# Método 1: Target CMake (recomendado)  
make run_tetris  
make run_snake  
make run_arkanoid  
  
# Método 2: Executar diretamente  
cd games/tetris && ./tetris  
cd games/snake && ./snake  
  
# Método 3: A partir do build/  
./games/tetris/tetris  
./games/snake/snake
```

## Opções de Build

### Tipos de Build

#### Debug (Padrão)

```
cmake .. -DCMAKE_BUILD_TYPE=Debug  
make all_games  
  
# Características:  
# - Símbolos de debug incluídos  
# - Otimizações desabilitadas  
# - Assertions habilitadas  
# - Executáveis maiores
```

## Release

```
cmake .. -DCMAKE_BUILD_TYPE=Release
make all_games
```

```
# Características:
# - Máxima otimização (-O3)
# - Sem símbolos de debug
# - Executáveis menores
# - Melhor performance
```

## RelWithDebInfo

```
cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
make all_games
```

```
# Características:
# - Otimizado + símbolos debug
# - Bom para profiling
# - Compromisso entre debug e performance
```

## Compilação Paralela

```
# Usar todos os cores disponíveis
make -j$(nproc) # Linux/macOS
make -j%NUMBER_OF_PROCESSORS% # Windows

# Ou definir permanentemente
export MAKEFLAGS="-j$(nproc)"
make all_games
```

## Compilação Verbose

```
# Ver comandos completos de compilação
make VERBOSE=1

# Ou configurar permanentemente
cmake .. -DCMAKE_VERBOSE_MAKEFILE=ON
make
```



## Configurações Avançadas

### Compilador Específico

```
# Usar GCC específico
cmake .. -DCMAKE_CXX_COMPILER=g++-9

# Usar Clang
cmake .. -DCMAKE_CXX_COMPILER=clang++

# Windows - Visual Studio
cmake .. -G "Visual Studio 16 2019"
```

## Flags de Compilação Customizadas

```
# Adicionar flags extras
cmake .. -DCMAKE_CXX_FLAGS="-Wall -Wextra -Wpedantic"

# Debug com sanitizers
cmake .. -DCMAKE_BUILD_TYPE=Debug \
        -DCMAKE_CXX_FLAGS="-fsanitize=address -fsanitize=undefined"
```

## Configurar SFML Manualmente

```
# Se SFML não for encontrado automaticamente
cmake .. -DSFML_ROOT=/usr/local/SFML

# Ou especificar bibliotecas
cmake .. -DSFML_LIBRARIES="sfml-system;sfml-window;sfml-graphics;sfml-audio"
```

## Estrutura de Build

### Diretórios Gerados

```
build/
├─ CMakeCache.txt           # Cache do CMake
├─ CMakeFiles/              # Arquivos internos do CMake
├─ Makefile                 # Makefile principal
├─ games/                   # Executáveis e assets
│   └─ tetris/
│       └─ tetris           # Executável
│           └─ images/      # Assets copiados
├─ snake/
│   └─ snake
│       └─ images/
```

```
|   └─ ... (para cada jogo)
└─ cmake_install.cmake    # Script de instalação
```

## Assets Automaticamente Gerenciados

O CMake copia automaticamente:

- `images/` → `build/games/<jogo>/images/`
- `fonts/` → `build/games/<jogo>/fonts/`
- `files/` → `build/games/<jogo>/files/`

## Scripts Úteis

### Build Script Personalizado

```
# Criar script de build personalizado
cat > quick_build.sh << 'EOF'
#!/bin/bash

GAME="$1"
BUILD_TYPE="${2:-Debug}"

if [ -z "$GAME" ]; then
    echo "Uso: $0 <jogo> [Debug|Release]"
    echo "Jogos: tetris, snake, arkanoid, etc."
    echo "Exemplo: $0 tetris Release"
    exit 1
fi

echo "🔨 Compilando $GAME em modo $BUILD_TYPE..."

# Criar/limpar build se necessário
if [ ! -d "build" ]; then
    mkdir build
fi

cd build

# Configurar se necessário
if [ ! -f "CMakeCache.txt" ] || [ "$BUILD_TYPE" != "$(cat CMakeCache.txt |
grep CMAKE_BUILD_TYPE | cut -d'=' -f2)" ]; then
    echo "⚙️ Configurando CMake..."
    cmake .. -DCMAKE_BUILD_TYPE="$BUILD_TYPE"
```

```

fi

# Compilar jogo específico
echo "🔧 Compilando..."
make "$GAME" -j$(nproc)

if [ $? -eq 0 ]; then
    echo "✅ $GAME compilado com sucesso!"
    echo "🎮 Para executar: make run_$GAME"
else
    echo "❌ Erro na compilação!"
    exit 1
fi
EOF

chmod +x quick_build.sh

# Usar o script
./quick_build.sh tetris Debug
./quick_build.sh snake Release

```

## Clean Build Script

```

# Script para limpeza completa
cat > clean_build.sh << 'EOF'
#!/bin/bash

echo "🧹 Limpando build anterior..."
rm -rf build

echo "📁 Criando diretório build..."
mkdir build
cd build

echo "⚙️ Configurando CMake..."
cmake ..

echo "🔧 Compilando todos os jogos..."
make all_games -j$(nproc)

echo "✅ Build limpo concluído!"
EOF

```

```
chmod +x clean_build.sh
./clean_build.sh
```



## Debug e Profiling

### Compilar para Debug

```
# Build com informações de debug
cmake .. -DCMAKE_BUILD_TYPE=Debug -DCMAKE_CXX_FLAGS="-g3 -O0"
make tetris

# Executar com GDB
gdb ./games/tetris/tetris
```

### Profiling com Valgrind

```
# Compilar com símbolos de debug
cmake .. -DCMAKE_BUILD_TYPE=Debug
make tetris

# Executar com Valgrind
valgrind --tool=memcheck --leak-check=full ./games/tetris/tetris
```

### Análise de Performance

```
# Compilar otimizado com símbolos
cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
make tetris

# Profiling com perf (Linux)
perf record ./games/tetris/tetris
perf report
```



## Troubleshooting de Compilação

### Erro: SFML não encontrado

```
# Verificar instalação
pkg-config --exists sfml-all
pkg-config --modversion sfml-all

# Limpar cache e reconfigurar
```

```
rm CMakeCache.txt
cmake .. -DSFML_ROOT=/usr/local
```

### Erro: Compilador não suporta C++17

```
# Verificar versão do compilador
g++ --version

# Usar compilador mais novo
cmake .. -DCMAKE_CXX_COMPILER=g++-9
```

### Erro: Assets não encontrados

```
# Verificar se assets foram copiados
ls build/games/tetris/images/

# Recompilar para forçar cópia
make clean
make tetris
```

### Erro: "make: command not found"

```
# Linux - instalar build-essential
sudo apt install build-essential

# macOS - instalar Xcode Command Line Tools
xcode-select --install

# Windows - usar cmake --build
cmake --build . --target all_games
```



## Monitoramento de Build

### Tempo de Compilação

```
# Medir tempo total
time make all_games

# Medir por jogo
time make tetris
```

### Uso de Recursos



```
# Monitor durante build
htop # Em outro terminal

# Compilação com limite de CPU
make -j2 all_games # Usar apenas 2 cores
```

## Tamanho dos Executáveis

```
# Ver tamanho de todos os jogos
du -sh games/*/

# Detalhes de um jogo específico
ls -lah games/tetris/tetris
file games/tetris/tetris
```

## Comandos de Referência Rápida

```
# Setup inicial
./setup.sh

# Compilar tudo
make all_games

# Compilar jogo específico
make <nome_do_jogo>

# Executar jogo
make run_<nome_do_jogo>

# Limpar e reconstruir
rm -rf build && mkdir build && cd build && cmake .. && make all_games

# Build otimizado
cmake .. -DCMAKE_BUILD_TYPE=Release && make all_games

# Build paralelo
make -j$(nproc) all_games

# Ver comandos disponíveis
make help
```

## Próximos Passos

Agora que você domina a compilação:

1. Explore para execução prática
2. Veja para mais opções de execução
3. Consulte para entender o código

Se encontrar problemas:

1. Verifique Troubleshooting ([Troubleshooting](#))
2. Confirme que seguiu Configuração do Ambiente ([Configuração do Ambiente](#))

Dica: Use `make -j$(nproc)` para compilação mais rápida, e `make run_<jogo>` para executar diretamente!

# Troubleshooting

Este guia resolve os problemas mais comuns encontrados durante a configuração e execução dos 16 jogos em C++. 🔧

## Problemas Mais Comuns

### 1. SFML não encontrado

#### Sintomas

```
CMake Error: Could not find SFML
pkg-config: sfml-all not found
```

#### Soluções

##### Linux:

```
# Ubuntu/Debian
sudo apt update
sudo apt install libsFML-dev

# Fedora
sudo dnf install SFML-devel

# Arch Linux
sudo pacman -S sfml

# Verificar instalação
pkg-config --exists sfml-all && echo "OK" || echo "ERRO"
```

##### macOS:

```
# Usando Homebrew
brew install sfml

# Se Homebrew não estiver instalado
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

##### Windows:

- Baixe SFML do site oficial (<https://www.sFML-dev.org/download.php>)

- Extraia para `C:\SFML`
- Configure variável de ambiente `SFML_ROOT=C:\SFML`

## 2. CMake versão muito antiga

### Sintomas

```
CMake Error: CMake 3.5 or higher is required. You are running version 2.8.12
```

### Soluções

#### Ubuntu/Debian:

```
# Remover versão antiga
sudo apt remove cmake

# Adicionar repositório oficial
wget -O - https://apt.kitware.com/keys/kitware-archive-latest.asc | sudo apt-
key add -
sudo apt-add-repository 'deb https://apt.kitware.com/ubuntu/ focal main'
sudo apt update
sudo apt install cmake
```

#### Compilar do código fonte:

```
wget https://github.com/Kitware/CMake/releases/download/v3.25.1/cmake-
3.25.1.tar.gz
tar -xzf cmake-3.25.1.tar.gz
cd cmake-3.25.1
./bootstrap --prefix=/usr/local
make -j$(nproc)
sudo make install
```

## 3. Compilador não suporta C++17

### Sintomas

```
error: 'auto' type specifier is a C++11 extension
error: range-based for loop is a C++11 extension
```

### Soluções

#### Ubuntu/Debian:

```
# Instalar GCC mais recente
sudo apt install gcc-9 g++-9
```

```
# Configurar como padrão
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-9 90
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-9 90

# Verificar versão
g++ --version
```

**Forçar compilador no CMake:**

```
cmake .. -DCMAKE_CXX_COMPILER=g++-9
```

## 4. Erro de linking com SFML

**Sintomas**

```
undefined reference to `sf::RenderWindow::RenderWindow()'
undefined reference to `sf::Texture::loadFromFile()'
```

**Soluções**

**Verificar bibliotecas SFML:**

```
# Linux
ldconfig -p | grep sfml
find /usr -name "*sfml*" 2>/dev/null

# Verificar pkg-config
pkg-config --cflags --libs sfml-all
```

**Reinstalar SFML:**

```
# Ubuntu/Debian
sudo apt remove libsfml-dev
sudo apt autoremove
sudo apt install libsfml-dev

# Verificar novamente
pkg-config --modversion sfml-all
```

## 5. Jogos não iniciam (sem janela)

**Sintomas**

- Executável compila mas não abre janela

- Erro "Failed to create OpenGL context"
- Tela preta

## Soluções

### Verificar drivers gráficos:

```
# Linux - informações da GPU
lspci | grep -i vga
glxinfo | grep "OpenGL version"

# Instalar drivers se necessário
# NVIDIA:
sudo apt install nvidia-driver-470

# AMD:
sudo apt install mesa-vulkan-drivers

# Intel:
sudo apt install intel-media-va-driver
```

### Testar OpenGL:

```
# Instalar mesa-utils
sudo apt install mesa-utils

# Testar OpenGL
glxgears
```

### Executar com debug:

```
# Executar com informações de debug
DISPLAY=:0 ./games/tetris/tetris
```

## 6. Assets não encontrados

### Sintomas

```
Failed to load image: images/tiles.png
Failed to load font: fonts/arial.ttf
```

## Soluções

### Verificar estrutura de arquivos:

```
# Ver se assets foram copiados
ls build/games/tetris/
ls build/games/tetris/images/

# Se não existirem, recompilar
make clean
make tetris
```

**Executar do diretório correto:**

```
# CORRETO - executar de dentro do diretório do jogo
cd build/games/tetris
./tetris

# INCORRETO - executar de outro lugar
cd build
./games/tetris/tetris # Pode não encontrar assets
```

## 7. Erro de permissão

**Sintomas**

```
Permission denied
make: *** [CMakeFiles/tetris.dir/all] Error 2
```

**Soluções**

**Corrigir permissões:**

```
# Dar permissão de execução aos scripts
chmod +x setup.sh
chmod +x *.sh

# Corrigir permissões do projeto
chmod -R 755 .
```

**Problemas de sudo:**

```
# Se instalou com sudo, corrigir ownership
sudo chown -R $USER:$USER ~/.cmake
sudo chown -R $USER:$USER ./build
```



## Problemas Específicos por Sistema

## Ubuntu/Debian Específicos

**Erro: "Package sfml-all was not found"**

```
# Atualizar lista de pacotes
sudo apt update

# Verificar se universe repository está habilitado
sudo add-apt-repository universe
sudo apt update

# Instalar SFML
sudo apt install libsFML-dev
```

**Erro: "Unable to locate package"**

```
# Verificar versão do Ubuntu
lsb_release -a

# Ubuntu muito antigo - usar PPA
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt update
```

## Fedora/CentOS Específicos

**Erro: "No package SFML-devel available"**

```
# Fedora - habilitar RPM Fusion
sudo dnf install https://mirrors.rpmfusion.org/free/fedora/rpmfusion-free-release-$(rpm -E %fedora).noarch.rpm

# CentOS - habilitar EPEL
sudo dnf install epel-release
```

## macOS Específicos

**Erro: "xcrun: error: invalid active developer path"**

```
# Instalar Command Line Tools
xcode-select --install

# Se já instalado, resetar
sudo xcode-select --reset
```

**Homebrew não funciona**



```
# Reinstalar Homebrew
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Adicionar ao PATH
echo 'export PATH="/opt/homebrew/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc
```

## Windows Específicos

### Visual Studio não encontra SFML

```
# No CMakeLists.txt, adicionar:
set(SFML_ROOT "C:/SFML")
find_package(SFML 2.5 COMPONENTS system window graphics audio REQUIRED)
```


### MinGW problemas de linking


```
# Usar bibliotecas estáticas
cmake .. -DSFML_STATIC_LIBRARIES=TRUE
```



## Ferramentas de Diagnóstico

### Script de Diagnóstico Completo

```
# Criar script de diagnóstico
cat > diagnose.sh << 'EOF'
#!/bin/bash

echo " Diagnóstico do Sistema - 16 Games in C++"
echo "===== "

# Sistema operacional
echo " Sistema:"
uname -a
echo ""

# Compilador
echo " Compilador:"
if command -v g++ &> /dev/null; then
    g++ --version | head -1
    echo " G++ disponível"
else
```

```

    echo "❌ G++ não encontrado"
fi

if command -v clang++ &> /dev/null; then
    clang++ --version | head -1
    echo "✅ Clang++ disponível"
else
    echo "❌ Clang++ não encontrado"
fi
echo ""

# CMake
echo "🔧 CMake:"
if command -v cmake &> /dev/null; then
    cmake --version | head -1
    echo "✅ CMake disponível"
else
    echo "❌ CMake não encontrado"
fi
echo ""

# SFML
echo "🎮 SFML:"
if pkg-config --exists sfml-all; then
    echo "✅ SFML $(pkg-config --modversion sfml-all) encontrado"
    echo "📌 Flags: $(pkg-config --cflags --libs sfml-all)"
else
    echo "❌ SFML não encontrado via pkg-config"

    # Procurar manualmente
    if find /usr -name "*sfml*" 2>/dev/null | head -5; then
        echo "💡 SFML pode estar instalado mas não configurado para pkg-
config"
    fi
fi
echo ""

# OpenGL
echo "🖥️ OpenGL:"
if command -v glxinfo &> /dev/null; then
    echo "OpenGL: $(glxinfo | grep "OpenGL version" | cut -d':' -f2)"
    echo "✅ OpenGL disponível"
else
    echo "⚠️ glxinfo não disponível (instale mesa-utils)"

```

```

fi
echo ""

# Espaço em disco
echo "💾 Espaço em disco:"
df -h . | tail -1
echo ""

# Resumo
echo "📋 Resumo:"
echo "======"

issues=0

if ! command -v g++ &> /dev/null && ! command -v clang++ &> /dev/null; then
    echo "❌ Nenhum compilador C++ encontrado"
    ((issues++))
fi

if ! command -v cmake &> /dev/null; then
    echo "❌ CMake não encontrado"
    ((issues++))
fi

if ! pkg-config --exists sfml-all; then
    echo "❌ SFML não encontrado"
    ((issues++))
fi

if [ $issues -eq 0 ]; then
    echo "🎉 Sistema parece estar configurado corretamente!"
    echo "💡 Se ainda há problemas, execute: ./setup.sh"
else
    echo "⚠️ $issues problema(s) encontrado(s)"
    echo "💡 Consulte a documentação para resolver os problemas acima"
fi
EOF

chmod +x diagnose.sh
./diagnose.sh

```

## Verificação de Build

```

# Script para verificar build específico
cat > check_build.sh << 'EOF'
#!/bin/bash

if [ ! -d "build" ]; then
    echo "❌ Diretório build não existe"
    echo "Execute: mkdir build && cd build && cmake .."
    exit 1
fi

cd build

if [ ! -f "CMakeCache.txt" ]; then
    echo "❌ CMake não foi configurado"
    echo "Execute: cmake .."
    exit 1
fi

echo "✅ Build configurado"
echo "📁 Jogos compilados:"

count=0
for game_dir in games/*/; do
    if [ -d "$game_dir" ]; then
        game_name=$(basename "$game_dir")
        if [ -f "$game_dir/$game_name" ]; then
            echo "✅ $game_name"
            ((count++))
        else
            echo "❌ $game_name (não compilado)"
        fi
    fi
done

echo ""
echo "📊 Total: $count jogos compilados"

if [ $count -eq 0 ]; then
    echo "💡 Execute: make all_games"
fi
EOF

```

```
chmod +x check_build.sh
./check_build.sh
```

## Últimos Recursos

### Resetar Ambiente Completamente

```
# Script de reset total
cat > reset_environment.sh << 'EOF'
#!/bin/bash

echo "🔥 RESETANDO AMBIENTE COMPLETAMENTE"
echo "===== "

# Fazer backup se necessário
if [ -d "build" ]; then
    echo "📦 Fazendo backup do build atual..."
    mv build build_backup_$(date +%Y%m%d_%H%M%S)
fi

# Limpar completamente
echo "🧹 Limpando arquivos temporários..."
rm -rf build
rm -rf .cache
find . -name "*.o" -delete
find . -name "*.cmake" -delete 2>/dev/null

# Recriar build
echo "📁 Recriando estrutura..."
mkdir build
cd build

# Configurar do zero
echo "⚙️ Configurando CMake do zero..."
cmake .. -DCMAKE_BUILD_TYPE=Debug

# Compilar teste
echo "🔨 Testando compilação..."
make tetris

if [ $? -eq 0 ]; then
    echo "✅ Reset concluído com sucesso!"
    echo "🎮 Teste: make run_tetris"
```

```
else
    echo "❌ Ainda há problemas após reset"
    echo "💡 Execute o diagnóstico: ../diagnose.sh"
fi
EOF

chmod +x reset_environment.sh
```

## Suporte da Comunidade

Se nenhuma solução funcionou:

1. Execute o diagnóstico completo: `./diagnose.sh`

2. Tente o reset total: `./reset_environment.sh`

3. Procure ajuda online:

- Stack Overflow: tag `sfml` + `cmake`
- Reddit: `r/cpp`, `r/gamedev`
- Discord: servidores de C++ e game dev

4. Documente seu problema:

- Sistema operacional e versão
- Saída do script de diagnóstico
- Mensagens de erro completas
- Passos que já tentou

## Prevenção de Problemas

### Manutenção Regular

```
# Atualizar dependências mensalmente
sudo apt update && sudo apt upgrade # Linux
brew update && brew upgrade          # macOS

# Limpar builds antigos
find . -name "build*" -type d -mtime +30 -exec rm -rf {} \;
```

### Backup de Configuração

```
# Backup da configuração funcionando
tar -czf working_config_$(date +%Y%m%d).tar.gz \
    CMakeLists.txt setup.sh build/CMakeCache.txt
```

**Lembre-se:** A maioria dos problemas pode ser resolvida com `./setup.sh`. Em caso de dúvida, sempre comece pelo diagnóstico automático!