IN204

Quoridor (C++/SFML) with Heuristic AI and Modern OOP Architecture

CHACÓN José Daniel

MENESES Carlos Adrián

École Nationale des techniques avancées

January 2026

Contents

1 Project Overview

The project implements a C++/SFML board game aiming to demonstrate object-oriented design, interactive rendering, and a CPU opponent based on search algorithms. The game emphasizes turn-based decision-making, rule validation, and a clear separation between game logic, rendering, and user interface management. The technology stack is C++ with SFML for graphics, input, and audio.

Main Objectives

- Implement a complete turn-based board game with a clear victory condition.
- Enforce the legality of moves and wall placements at runtime.
- Provide a responsive graphical interface with screen-based navigation.
- Integrate a heuristic CPU player based on search and evaluation.
- Maintain a modular codebase with well-defined responsibilities.

2 Game Presentation: What is Quoridor?

The game is a two-player strategy game played on a grid. Each player controls a pawn and attempts to reach the opposite edge of the board. On each turn, a player either moves their pawn or places a wall to modify available paths. The objective is to reach the target row before the opponent. Wall placements must preserve at least one valid path to the objective for each player.

3 Game Mechanics (System Level)

At startup, the application initializes window resources, loads assets, and constructs the initial game state. The main loop follows a standard input–update–render cycle. Input is captured via SFML events and routed to the active screen. The update step advances animations, manages turn logic, and triggers CPU calculation when necessary. The render step draws the board, interface elements, and screen-specific components.

A screen system manages different contexts such as title, menu, gameplay, and credits. The game screen coordinates the rules, visual board, and heuristic engine. A bottom-screen HUD can display the current turn and remaining walls. A pause menu is accessible via a keyboard shortcut and allows resuming or restarting the game.

4 Technical Stack

Language and Libraries

The project uses C++20 and the SFML library for graphics, input, and audio. SFML provides the rendering window, sprite management, and event handling necessary for the game loop.

Build System

The compilation process is managed with CMake (minimum version **3.22**). The project targets SFML version **3.0**. The build configuration can be adapted for Windows (MSVC) or Linux (g++/clang).

5 Project Structure

Directory Tree

```
<PROJECT_NAME>/
|-- CMakeLists.txt
|-- README.md
|-- LICENSE
|-- docs/
|   `-- Initial Report/
|-- include/
|   |-- app/
|   |-- game/
|   |-- heuristic/
|   `-- ui/
|-- src/
|   |-- app/
|   |-- game/
|   |-- heuristic/
|   |-- ui/
|   `-- main.cpp
`-- assets/
    |-- fonts/
    |-- sound/
    `-- textures/
```

Role of Directories

- `src/` : implementation files for application flow, gameplay, AI, and UI.
- `include/` : public headers defining main interfaces and structures.
- `assets/` : textures, fonts, and audio used at runtime.
- `Docs/` : project documentation and reports.

Main Classes/Modules

The architecture includes a base screen class and derived screens such as `TitleScreen`, `GameScreen`, and `CreditsScreen`. Game logic is encapsulated in a dedicated rules module, while a UI module handles rendering, HUD elements, and menus. The heuristic engine evaluates positions and selects CPU moves via bounded search.

6 Installation and Execution

Prerequisites

- C++20 compiler (MSVC, g++, or clang)
- CMake **3.22** or later
- SFML **3.0**
- Git
- Linux development libraries (for Linux builds):

```
sudo apt install -y \
  build-essential git \
  libx11-dev libxrandr-dev libxcursor-dev libxi-dev \
  libudev-dev \
  libgl1-mesa-dev \
  libfreetype-dev \
  libvorbis-dev libflac-dev
```

Clone Repository

```
git clone https://github.com/josedanielchg/quoridor-strategy-game.git
cd quoridor-strategy-game
```

Compilation (Windows)

```
cmake -S . -B build
cmake --build build
```

Compilation (Linux)

```
cmake -S . -B build
cmake --build build
```

Execution (Windows)

```
.\build\Debug\QuoridorGame.exe
```

Execution (Linux)

```
./build/QuoridorGame
```