



IN204

# Quoridor (C++/SFML) avec IA Heuristique et Architecture POO Moderne

CHACÓN José Daniel  
MENESES Carlos Adrián

École Nationale des techniques avancées  
Janvier 2026

## Table des matières

---

<b>1 Aperçu du projet</b>	<b>2</b>
<b>2 Présentation du jeu : qu'est-ce que “Corridor” ?</b>	<b>2</b>
<b>3 Fonctionnement du jeu (niveau système)</b>	<b>2</b>
<b>4 Pile technique</b>	<b>2</b>
<b>5 Structure du projet</b>	<b>3</b>
<b>6 Installation et exécution</b>	<b>4</b>

## 1 Aperçu du projet

---

Le projet implémente un jeu de plateau C++/SFML visant à démontrer la conception orientée objet, le rendu interactif et un adversaire CPU basé sur la recherche. Le jeu met l'accent sur la prise de décision au tour par tour, la validation des règles et une séparation claire entre la logique de jeu, le rendu et la gestion de l'interface utilisateur. La pile technologique est C++ avec SFML pour les graphismes, l'entrée et l'audio.

### Objectifs principaux

- Implémenter un jeu de plateau complet au tour par tour avec une condition de victoire claire.
- Imposer la légalité des déplacements et des placements de murs à l'exécution.
- Fournir une interface graphique réactive avec une navigation par écrans.
- Intégrer un joueur CPU heuristique basé sur la recherche et l'évaluation.
- Maintenir une base de code modulaire avec des responsabilités bien définies.

## 2 Présentation du jeu : qu'est-ce que “Corridor” ?

---

Le jeu est un jeu de stratégie à deux joueurs joué sur une grille. Chaque joueur contrôle un pion et tente d'atteindre le bord opposé du plateau. À chaque tour, un joueur déplace son pion ou place un mur pour modifier les chemins disponibles. L'objectif est d'atteindre la rangée cible avant l'adversaire. Le placement des murs doit préserver au moins un chemin valide vers l'objectif pour chaque joueur.

## 3 Fonctionnement du jeu (niveau système)

---

À l'exécution, l'application initialise les ressources de la fenêtre, charge les assets et construit l'état initial du jeu. La boucle principale suit un cycle standard entrée–mise à jour–rendu. Les entrées sont capturées via les événements SFML, puis routées vers l'écran actif. L'étape de mise à jour fait avancer les animations, gère la logique de tour et déclenche le calcul CPU lorsque nécessaire. L'étape de rendu dessine le plateau, les éléments d'interface et les composants propres à chaque écran.

Un système d'écrans gère différents contextes tels que le titre, le menu, le gameplay et les crédits. L'écran de jeu coordonne les règles, le plateau visuel et le moteur heuristique. Un HUD en bas d'écran peut afficher le tour en cours et les murs restants. Un menu pause est accessible via un raccourci clavier et permet de reprendre ou de redémarrer la partie.

## 4 Pile technique

---

### Langage et bibliothèques

Le projet utilise C++20 et la bibliothèque SFML pour les graphismes, les entrées et l'audio. SFML fournit la fenêtre de rendu, la gestion des sprites et la collecte d'événements nécessaires à la boucle de jeu.

## Système de build

Le processus de compilation est géré avec CMake (version minimale **3.16**). Le projet cible SFML version **3.0**. La configuration de build peut être adaptée pour Windows (MSVC) ou Linux (g++/clang).

## 5 Structure du projet

---

### Arborescence

```
<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/
```

### Rôle des dossiers

- **src/** : fichiers d'implémentation pour le flux applicatif, le gameplay, l'IA et l'UI.
- **include/** : en-têtes publics définissant les interfaces et structures principales.
- **assets/** : textures, polices et audio utilisés à l'exécution.
- **Docs/** : documentation et rapports du projet.

### Classes/modules principaux

L'architecture inclut une classe de base d'écran et des écrans dérivés tels que **<ScreenTitle>**, **<ScreenGame>** et **<ScreenCredits>**. La logique de jeu est encapsulée dans un module de règles dédié, tandis qu'un module UI gère le rendu, les éléments de HUD et les menus. Le moteur heuristique évalue les positions et sélectionne les coups CPU via une recherche bornée.

## 6 Installation et exécution

---

### Prérequis

- Compilateur C++20 (MSVC, g++ ou clang)
- CMake **3.16** ou plus récent
- SFML **3.0**
- Git

### Cloner le dépôt

```
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
```

### Exécution (Windows)

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

### Exécution (Linux)

```
./build/QuoridorGame
```