Planification du projet Quoridor (C++/SFML)

Développement en 4 itérations

Équipe: Tarazona Javier, Liang Tianyi

17 novembre 2025 – 15 janvier 2026

Vue d'ensemble

Le projet Quoridor sera développé selon une méthodologie itérative et incrémentale, avec une soutenance (défense) prévue le **15 janvier 2026**. Ce document planifie quatre itérations de développement, chacune produisant un incrément fonctionnel et testable du jeu.

Objectif final

Jeu de Quoridor 2D jouable (modes Humain vs Humain et Humain vs IA), architecture MVC claire, IA configurable (Facile/Normal/Difficile), exécutables cliquables multi-plateformes (Windows/Linux/macOS), tests unitaires, documentation et présentation de soutenance.

Calendrier global

- **Itération 1** : 18 novembre – 1 décembre 2025 (2 semaines)
- **Itération 2** : 2 décembre – 15 décembre 2025 (2 semaines)
- **Itération 3** : 16 décembre – 5 janvier 2026 (3 semaines, incluant vacances)
- **Itération 4** : 6 janvier – 14 janvier 2026 (1 semaine + préparation soutenance)
- **Soutenance** : 15 janvier 2026

Itération 1 : Fondations et prototype minimal

Dates : 18 novembre – 1 décembre 2025 (2 semaines)

Objectifs

- Mise en place de l'environnement de développement (CMake, SFML, Git).
- Architecture MVC de base fonctionnelle.
- Plateau 9×9 affiché avec SFML.
- Déplacement de pions (validation simple).
- Tests unitaires pour règles de déplacement.

Livrables

L1.1. **Infrastructure** :

- `CMakeLists.txt` fonctionnel (compilation, link SFML).
- Structure de dossiers (`src/`, `include/`, `assets/`, `tests/`).
- Dépôt Git avec branches et commits réguliers.

L1.2. **Modèle** :

- Classe `Board` : représentation grille 9×9, positions pions.
- Classe `State` : état du jeu (positions, joueur actif).
- Classe `Rules` (partielle) : validation déplacements orthogonaux simples.

L1.3. **Vue** :

- Classe `Renderer2D` : affichage grille, pions (formes géométriques ou sprites simples).
- Fenêtre SFML 800×600 pixels.

L1.4. **Contrôleur** :

- Classe `Game` : boucle principale (poll événements, update, render).
- Classe `InputHandler` : détection clics souris sur cases.

L1.5. **Tests** :

- Framework de tests intégré (Catch2 ou équivalent).
- Tests unitaires pour `Rules::isValidMove` (déplacements de base).

L1.6. **Fonctionnalité démo** : Déplacer un pion par clics successifs (case source, case cible), alternance joueurs.

Critères de succès

- Compilation sans erreur sur Windows.
- Fenêtre SFML affiche plateau 9×9 et 2 pions.
- Déplacement de pions validé et reflété visuellement.
- Au moins 5 tests unitaires passent (déplacements valides/invalides).

Risques et mitigations

- *Risque* : Difficultés installation SFML.
- *Mitigation* : Utiliser vcpkg ou Conan ; prévoir 2–3 jours pour setup environnement.

Itération 2 : Règles complètes et placement de murs

Dates : 2 décembre – 15 décembre 2025 (2 semaines)

Objectifs

- Implémenter règles complètes de déplacement (sauts par-dessus adversaire).
- Placement de murs avec validation (chevauchement, limites).
- Vérification chemin restant (BFS ou A*).
- Condition de victoire.
- Interface améliorée (affichage murs, HUD basique).

Livrables

L2.1. **Modèle** :

- `Board` : gestion murs (horizontaux/verticaux), stock murs par joueur.
- `Rules` (complet) : déplacements avec sauts, validation placement murs, interdiction blocage complet.
- `Pathfinder` : BFS pour vérifier existence chemin vers ligne d'arrivée.

L2.2. **Vue** :

- Rendu murs (rectangles colorés).
- HUD : affichage joueur actif, murs restants.
- Messages d'erreur (coup invalide).

L2.3. **Contrôleur** :

- `InputHandler` : mode placement mur (sélection position + orientation via clavier/souris).
- Basculement déplacement/placement mur (touche ou bouton UI).

L2.4. **Tests** :

- Tests placement murs (validations, chevauchements).
- Tests sauts (configurations adversaire bloquant).
- Tests pathfinding (cas limites : 1 seul chemin, plusieurs chemins).

L2.5. **Fonctionnalité démo** : Partie Humain vs Humain complète jusqu'à victoire.

Critères de succès

- Partie jouable de A à Z en mode local (2 humains).
- Victoire détectée et affichée.
- Tous les cas de règles validés par tests unitaires (>20 tests).
- Aucun placement de mur ne peut bloquer définitivement un joueur.

Risques et mitigations

- *Risque* : Complexité validation murs + pathfinding.
- *Mitigation* : Implémenter BFS simple d'abord ; tester cas par cas ; prévoir debug intensif.

Itération 3 : Intelligence artificielle et configurabilité

Dates : 16 décembre 2025 – 5 janvier 2026 (3 semaines)

Objectifs

- Implémenter IA Minimax/Négamax avec élagage α–β.
- Heuristiques pour évaluation positions (distance à l'arrivée, murs adversaire).
- Trois niveaux de difficulté (Facile/Normal/Difficile).
- Mode Humain vs IA fonctionnel.
- Amélioration UI/UX (menus, animations simples).

Livrables

L3.1. **Modèle** :

- Classe `AI` : Minimax/Négamax + élagage α–β, profondeur paramétrable.
- Heuristique : combinaison distance joueur/adversaire (A*), valeur murs.
- Randomisation légère pour coups de score proche.
- `Pathfinder` : A* pour estimation distance (optimisation heuristique IA).

L3.2. **Vue** :

- Écran menu : sélection mode (HvH, HvIA), difficulté IA.

— Feedback visuel coup IA (surlignage case/mur joué).
— Animation simple déplacement pions (optionnel).

L3.3. **Contrôleur** :
— `SceneManager` : gestion états (Menu, Partie, Pause, Fin).
— `Config` : chargement paramètres (fichier JSON ou constantes) : difficulté IA, résolution fenêtre.
— Tour IA : appel `AI::getBestMove()`, application coup, update Vue.

L3.4. **Tests** :
— Tests génération coups valides par IA.
— Tests heuristique (scores positions connues).
— Tests Minimax (profondeur 1–2, résultats attendus).

L3.5. **Fonctionnalité démo** : Partie Humain vs IA (difficulté Normal), IA joue en <2 secondes par coup.

Critères de succès

— IA niveau Normal bat un joueur occasionnel.
— IA niveau Difficile termine partie en <30 coups contre joueur moyen.
— IA niveau Facile commet erreurs visibles (profondeur faible, randomisation forte).
— Temps réponse IA acceptable (<3 secondes par coup, profondeur 3–4).

Risques et mitigations

— *Risque* : Performance IA trop lente (explosion combinatoire).
— *Mitigation* : Limiter profondeur (3–4) ; élagage α–β strict ; optimiser génération coups ; cache positions si temps disponible.
— *Risque* : Période vacances (moins de disponibilité).
— *Mitigation* : Planifier tâches modulaires ; commits réguliers ; travail asynchrone possible.

Itération 4 : Packaging, polish et préparation soutenance

Dates : 6 janvier – 14 janvier 2026 (1 semaine)

Objectifs

— Packaging multi-plateforme (Windows ZIP, macOS DMG/ZIP, Linux AppImage/tar.gz).
— Polissage UI (textures/sprites, polices, sons optionnels).
— Documentation complète (README, guide utilisateur, commentaires code).
— Préparation diaporama soutenance.
— Tests finaux et corrections bugs.

Livrables

L4.1. **Packaging** :
— `CMakeLists.txt` avec installation assets, copie DLL (Windows), bundle macOS.
— CPack configuré : générateurs ZIP (Windows), DragNDrop/ZIP (macOS), TGZ (Linux).
— Script packaging AppImage (Linux) ou alternative tar.gz + launcher.

— Artefacts finaux : `Quoridor-1.0-Windows.zip`, `Quoridor-1.0-macOS.dmg`, `Quoridor-1.0-Linux`

L4.2. **Assets et polish** :

— Textures/sprites pour plateau, pions, murs (ou maintien formes géométriques épurées).
— Police lisible pour HUD et messages.
— Icône application (Windows .ico, macOS .icns).
— Sons optionnels (déplacement, placement mur, victoire).

L4.3. **Documentation** :

— `README.md` : description, prérequis, compilation, exécution, modes de jeu.
— Guide utilisateur (PDF ou Markdown) : règles Quoridor, commandes, captures d'écran.
— Commentaires code (Doxygen-style ou minimal clair).
— Diagramme classes UML (optionnel mais recommandé pour soutenance).

L4.4. **Tests finaux** :

— Tests intégration : scénarios partie complète (HvH, HvIA).
— Tests plateformes : vérifier exécutables Windows/macOS/Linux (si accès machines).
— Corrections bugs identifiés.

L4.5. **Soutenance** :

— Diaporama (15–20 slides) : contexte, architecture MVC, IA (Minimax/A*), démo, résultats tests, packaging, rétrospective.
— Démo live : partie HvIA niveau Difficile, montrer UI/animations.
— Répétition orale (timing 10–15 minutes + questions).

Critères de succès

— Exécutables cliquables fonctionnels sur au moins 2 plateformes (Windows + macOS ou Linux).
— README complet permet à un utilisateur externe de compiler et jouer.
— Soutenance fluide, démo sans crash, questions anticipées.
— Code source propre, commenté, tests >90% passent.

Risques et mitigations

— *Risque* : Bugs de dernière minute.
— *Mitigation* : Geler fonctionnalités le 10 janvier ; jours 11–14 janvier réservés bug fixes et répétitions.
— *Risque* : Packaging complexe (macOS bundle, AppImage).
— *Mitigation* : Prioriser Windows ZIP (simple) ; macOS/Linux en best-effort ; documenter limitations si nécessaire.

Tableau récapitulatif des jalons

Itération	Dates	Jalons clés
Itération 1	18 nov – 1 déc	Infrastructure CMake/SFML, plateau affiché, déplacements pions simples, tests unitaires de base.
Itération 2	2 déc – 15 déc	Règles complètes (sauts, murs, pathfinding BFS), victoire détectée, partie HvH jouable.
Itération 3	16 déc – 5 jan	IA Minimax + α–β, 3 niveaux difficulté, mode HvIA, menus, A* pour heuristique.
Itération 4	6 jan – 14 jan	Packaging multi-plateforme, polish UI/assets, documentation complète, préparation soutenance.
Soutenance	15 janvier 2026	Présentation + démo + questions.

Recommandations de gestion de projet

Pratiques de développement

- **Git** : Commits fréquents (au moins quotidiens), messages descriptifs. Branches pour features majeures (ex : `feature/ai`, `feature/walls`).
- **Revue de code** : Relecture croisée avant merge vers `main`. Pair programming pour composants critiques (IA, pathfinding).
- **Tests** : Écrire tests avant ou en parallèle du code (TDD light). Objectif couverture >80% pour `Model`.
- **Intégration continue (optionnel)** : GitHub Actions pour build automatique multi-plateforme + tests (si temps disponible).

Communication et coordination

- **Réunions hebdomadaires** : Point équipe (30 min) début de semaine : bilan itération, répartition tâches.
- **Kanban/Trello** : Board avec colonnes *À faire*, *En cours*, *Terminé*, *Bloqué*.
- **Documentation** : Wiki ou `docs/` pour décisions architecture, API internes, FAQ technique.

Gestion des risques généraux

- **Scope creep** : Éviter ajout fonctionnalités non essentielles (ex : réseau, replay sophistiqué) durant itérations 1–3. Liste *nice-to-have* pour post-soutenance.
- **Indisponibilité membre** : Répartition tâches modulaires ; documentation claire pour reprise travail.
- **Problèmes techniques bloquants** : Timeboxing (max 4h recherche) ; escalade vers enseignant/forum si non résolu.

Conclusion

Cette planification en 4 itérations assure une progression incrémentale du projet Quoridor, avec des jalons vérifiables et une montée en complexité maîtrisée. Chaque itération produit un livrable fonctionnel, permettant tests et ajustements continus. La semaine finale est dédiée au polish et à la préparation soutenance, maximisant les chances de présentation réussie le 15 janvier 2026.

Prochaines étapes immédiates (semaine du 18 novembre) :

1. Créer dépôt Git et structure dossiers.
2. Installer SFML (via vcpkg ou Conan).
3. Écrire `CMakeLists.txt` minimal et tester compilation.
4. Implémenter classes `Board`, `State`, `Renderer2D` (squelettes).
5. Afficher plateau vide dans fenêtre SFML.