



Rapport Projet IA

Jeu Quridor avec MiniMax

Réalisé par :

Ghilas Tidjet
Badredine Bouamama

Université Paris Cité

2024/2025

Table des matières

1	Objectifs du projet	4
1.1	Contexte pédagogique	4
1.2	Exigences du projet	4
1.3	Choix du jeu : Quoridor	4
1.4	Équipe de développement	5
2	Présentation du jeu Quoridor	6
2.1	Description générale	6
2.2	Règles du jeu	6
3	Organisation du projet et structure des fichiers	7
3.1	launcher.py	7
3.2	quoridor_menus.py	7
3.3	Projet IA.py	7
3.4	Bibliothèques et paradigmes utilisés	8
4	Guide d'utilisation du jeu Quoridor	10
4.1	Lancement du jeu	10
4.2	Interface utilisateur	11
4.2.1	Menu principal	11
4.2.2	Écran de plateau de jeu et interactions	11
4.3	Choix du niveau de l'IA	13
4.3.1	Mode « Joueur vs IA »	13
4.3.2	Mode « IA vs IA »	14
4.3.3	Mode « Batch IA vs IA »	15
5	Explication des principales classes et fonctions	16
5.1	Classe QuoridorMenuApp (dans quoridor_menus.py)	16
5.2	Exception ReturnToMenu (dans Projet IA.py)	16
5.3	Fonction launch_game(game_params)	16
5.4	Boucles principales de jeu	17
5.5	IA et évaluation	17
6	Analyse des heuristiques dans les trois fonctions d'évaluation	18
6.1	IA Facile (evaluer_position)	18
6.2	IA Intermédiaire (evaluer_position_intermediaire)	18
6.3	IA Difficile (evaluer_position_difficile)	19
6.4	Comparaison des approches	20
7	Évaluation expérimentale des IA : résultats finaux	21
7.1	Protocole de test	21
7.2	Analyse du nombre de victoires IA vs IA	21
7.2.1	Tableau de synthèse des résultats	21
7.2.2	Analyse des données	22
7.3	Analyse du nombre moyen de coups par partie IA vs IA	22
7.3.1	Tableau de synthèse	23
7.3.2	Interprétabilité du tableau de synthèse	23

8 Bilan du projet

25

Table des figures

1	Arborescence du projet	7
2	le message affiché par terminal après l'exécution de la commande	10
3	Écran d'accueil de jeu Quorridor	11
4	Écran de plateau de jeu	12
5	Sélection de la difficulté en mode Joueur vs IA.	13
6	Choix des difficultés pour IA vs IA.	14
7	Écran de fin de partie, annonce du vainqueur et options « Menu Principal » / « Rejouer ».	14
8	Configuration du nombre de matchs en mode Batch IA vs IA	15
9	Exemple de sortie console après 100 parties en mode batch	15
10	Exemple de sortie UI après 100 parties en mode batch	15

1 Objectifs du projet

1.1 Contexte pédagogique

Dans le cadre de l'Unité d'Enseignement « Intelligence Artificielle », nous avons réalisé un projet en binôme consistant à concevoir un jeu stratégique dans lequel un joueur humain affronte une Intelligence Artificielle. Ce projet a pour but de mettre en œuvre des algorithmes d'IA, à travers l'implémentation d'au moins trois niveaux de difficulté, allant d'une stratégie simple à une stratégie optimale. L'objectif est également d'analyser les performances des différentes IA à l'aide de tournois automatisés, permettant une comparaison claire des stratégies mises en place. Le jeu choisi pour ce projet est **Quoridor**, un jeu de stratégie à deux joueurs, à informations parfaites et sans hasard, ce qui en fait un excellent candidat pour l'étude et l'application d'algorithmes d'intelligence artificielle.

1.2 Exigences du projet

Le cahier des charges imposait plusieurs éléments structurants :

- **Choix d'un jeu stratégique** à deux joueurs, déterministe, à information parfaite, suffisamment complexe pour exiger une véritable stratégie, mais réalisable dans le temps imparti.
- **Développement d'au moins trois IA** de difficultés différentes, intégrant l'algorithme Minimax, avec élagage alpha-bêta, et différentes heuristiques d'évaluation.
- **Maintien d'un temps de réponse raisonnable**, afin d'assurer une expérience utilisateur fluide et agréable.
- **Organisation de tournois entre IA**, avec au moins 50 parties par duel, pour analyser les performances relatives des différentes stratégies.
- **Rédaction d'un rapport complet** contenant une description du jeu, des algorithmes utilisés, des choix de conception, une analyse des résultats, ainsi qu'un bilan du projet.

1.3 Choix du jeu : Quoridor

Notre projet consiste à développer une interface de jeu dotée de trois intelligences artificielles de niveaux de difficulté croissants. Après avoir étudié plusieurs jeux de société à portée algorithmique (Dames, Othello, Puissance 4, Quoridor), notre choix s'est finalement porté sur Quoridor pour les raisons suivantes :

- **Originalité et moindre fréquentation académique** Quoridor est peu exploité dans les projets étudiants, ce qui garantit un sujet plus original et, de fait, une évaluation moins standardisée et donc souvent moins sévère.
- **Simplicité des règles, richesse stratégique** Les règles de Quoridor sont simples

à formaliser (déplacement d'un pion et pose de barrières), tandis que la recherche du chemin le plus court confère une profondeur tactique et stratégique intéressante.

- **Complexité algorithmique modulable** Le problème principal (recherche de chemin) se prête naturellement à des techniques classiques (BFS, A, Minimax, élagage alpha-bêta) et autorise l'ajout de niveaux de difficulté via des heuristiques ou des recherches plus ou moins profondes.
- **Faisabilité de l'implémentation en Python** La structure du plateau (9×9 cases) et la simplicité du modèle de données facilitent la représentation du jeu et l'intégration rapide de l'IA, tout en conservant des performances interactives satisfaisantes.
- **Interface et visualisation conviviales** Quoridor se prête bien à une représentation graphique simple (cases et murs), ce qui nous permet de concentrer nos efforts sur l'optimisation des algorithmes plutôt que sur la complexité de l'UI.
- **Adaptabilité des niveaux de difficulté** Le jeu offre un cadre naturel pour différencier trois IA : une IA de niveau débutant , une IA moyenne et une IA avancée, garantissant ainsi une progression pédagogique cohérente.

1.4 Équipe de développement

Ce projet a été conçu et développé par :

- **TIDJET Ghilas**
- **BOUAMAMA Badredine**

2 Présentation du jeu Quoridor

2.1 Description générale

Quoridor est un jeu de stratégie abstrait conçu pour deux ou quatre joueurs. Dans notre projet, nous avons choisi la version à deux joueurs. Le but du jeu est d'être le premier à atteindre la ligne opposée du plateau.

2.2 Règles du jeu

Le jeu se joue sur un plateau de 9x9 cases. Chaque joueur commence au centre de sa ligne de départ (en bas pour le joueur 1, en haut pour le joueur 2). À chaque tour, un joueur peut :

- Déplacer son pion d'une case (haut, bas, gauche ou droite) si la case n'est pas bloquée ;
- Placer un mur (horizontal ou vertical) entre deux cases pour ralentir l'adversaire, sans bloquer totalement l'accès à la ligne d'arrivée.

Chaque joueur dispose d'un nombre limité de murs (10 par joueur). Il est interdit de placer un mur bloquant totalement le passage de l'adversaire. Le jeu demande une gestion stratégique des murs et des déplacements pour optimiser son propre parcours tout en gênant l'adversaire.

3 Organisation du projet et structure des fichiers

QUORRIDOR-GAME/	
__pycache__/	% Fichiers compilés Python
NovaSquare-Regular.ttf	% Ressource : police personnalisée
launcher.py	% Point d'entrée principal (transition Tk-Pygame)
quoridor_menus.py	% Interface Tkinter (menus de sélection)
Projet IA.py	% Mécanique de jeu (Pygame + IA + batch)

FIGURE 1 – Arborescence du projet

3.1 launcher.py

- **Responsabilité** : Point d'entrée principal qui enchaîne le menu Tkinter et le moteur Pygame.
- `main()` :
 1. Appelle `launch_menu()` (depuis `quoridor_menus.py`).
 2. Ferme proprement la fenêtre Tkinter.
 3. Invoque `quoridor_game.launch_game(game_params)`.

3.2 quoridor_menus.py

- **Responsabilité** : Interface graphique de sélection des modes de jeu, implémentée avec Tkinter.
- **Classe QuoridorMenuApp** :
 - Construit les menus : *Joueur vs Joueur*, *Joueur vs IA*, *IA vs IA*, *Batch IA vs IA*.
 - Convertit chaque choix en un dictionnaire `game_params` contenant :


```
{ mode, difficulty, ai1_difficulty, ai2_difficulty, num_matches }.
```
- `launch_menu()` : instancie `QuoridorMenuApp`, lance la boucle Tk et renvoie `game_params`.

3.3 Projet IA.py

- **Responsabilité** : Cœur du jeu Quoridor, implémenté avec Pygame, contient la logique de plateau, les IA et les simulations par lot.
- **Entrée principale** `launch_game(game_params)` :
 1. Initialise la fenêtre Pygame et charge la police.
 2. Sélectionne la routine à exécuter selon `game_params.mode` :

- `mainPVP()` — Partie joueur contre joueur.
- `mainPVE(difficulty)` — Partie joueur contre IA.
- `mainAIvsAI(d1,d2)` — Duel IA vs IA.
- `run_batch_simulations(d1,d2,n)` — Simulation de n matchs IA vs IA.
- **Mécanique du plateau :**
 - `creer_grille()`, `dessiner_grille()`, `dessiner_murs()`.
 - Gestion du clic et du hover : `gestion_clic_souris()`, `gestion_hover_souris()`.
- **Règles de déplacement et de mur :**
 - `get_possible_moves()`, `mouvement_est_valide()`, `mur_bloque_mouvement()`, `mur_est_valide()`.
- **Intelligences artificielles :**
 - Heuristiques :
 - `evaluer_position()` : fonction d'évaluation utilisée pour le niveau facile.
 - `evaluer_position_intermediaire()` : fonction utilisée pour le niveau intermédiaire.
 - `evaluer_position_difficile()` : fonction pour le niveau difficile.
 - Algorithmes de chemin : `a_star_search()`, `count_chemins_alternatifs()`.
 - Minimax α - β : `minimax()`, aides au choix de coup : `meilleur_coup_ia()`, `meilleur_deplacement_pour_joueur()`, `meilleur_mur_pour_joueur()`.
- **Simulations batch IA vs IA :**
 - `simulate_ai_vs_ai(d1,d2)` — simule un unique match sans UI.
 - `run_batch_simulations(d1,d2,n)` — lance n simulations, imprime les statistiques en console.

3.4 Bibliothèques et paradigmes utilisés

- **Langage :** Python 3.x
- **Interfaces graphiques :**
 - Tkinter (standard) pour les menus *desktop* (sélection de mode, difficulté, batch).
 - Pygame pour le rendu du plateau, la gestion des événements souris/clavier et les animations de jeu.
- **Modules standards :**
 - `sys`, `os`, `importlib.util` pour la découverte dynamique des scripts.
 - `collections.deque`, `heapq` pour les algorithmes de parcours de graphe.
 - `random` pour la génération de nombre aléatoires pour le facteur aléatoire dans l'IA facile et la génération de simulations variées.
- **Ressources externes :**
 - Police TrueType NovaSquare-Regular.ttf (chargée via `pygame.font` et `tkinter.font`).

- **Paradigmes et styles :**
 - *Impératif / modulaire* : code organisé en fonctions pures et procédures event-driven.
 - *Séparation des responsabilités* :
 - `launcher.py` (orchestration), `quoridor_menus.py` (UI Tkinter), `Projet IA.py` (logique du jeu).
 - *Algorithmes de recherche* :
 - A* (`a_star_search`) pour calculer les plus courts chemins.
 - Minimax avec élagage α - β (`minimax`) pour l'IA de niveau difficile.
 - *Heuristiques différenciées selon la difficulté* : trois fonctions d'évaluation (`evaluer_*`) graduant pondérations, aléa et profondeur de recherche.
 - *Programmation événementielle* : boucle principale Pygame (`while True + pygame.event.get()`), callbacks Tkinter.
 - *Simulation sans UI* : batch IA vs IA via fonctions pures (`simulate_ai_vs_ai`, `run_batch_simulations`).

4 Guide d'utilisation du jeu Quoridor

4.1 Lancement du jeu

Pour lancer l'application Quoridor, ouvrez PowerShell dans le répertoire contenant le script `start_quoridor.ps1`, puis exécutez la commande suivante :

```
1 .\start_quoridor.ps1
```

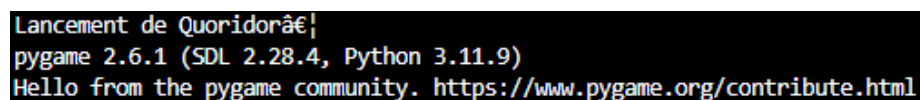
Listing 1 – Lancement de Quoridor via PowerShell

Le terminal affichera alors un message similaire à :

Lancement de Quoridor...

pygame 2.6.1 (SDL 2.28.4, Python 3.11.9)

Hello from the pygame community. <https://www.pygame.org/contribute.html>



```
Lancement de Quoridor!  
pygame 2.6.1 (SDL 2.28.4, Python 3.11.9)  
Hello from the pygame community. https://www.pygame.org/contribute.html
```

FIGURE 2 – le message affiché par terminal après l'exécution de la commande

et la fenêtre du jeu se lancera automatiquement après ces lignes.

4.2 Interface utilisateur

4.2.1 Menu principal

Au lancement (`python run.py`), la fenêtre Pygame affiche l'écran d'accueil ci-dessous :

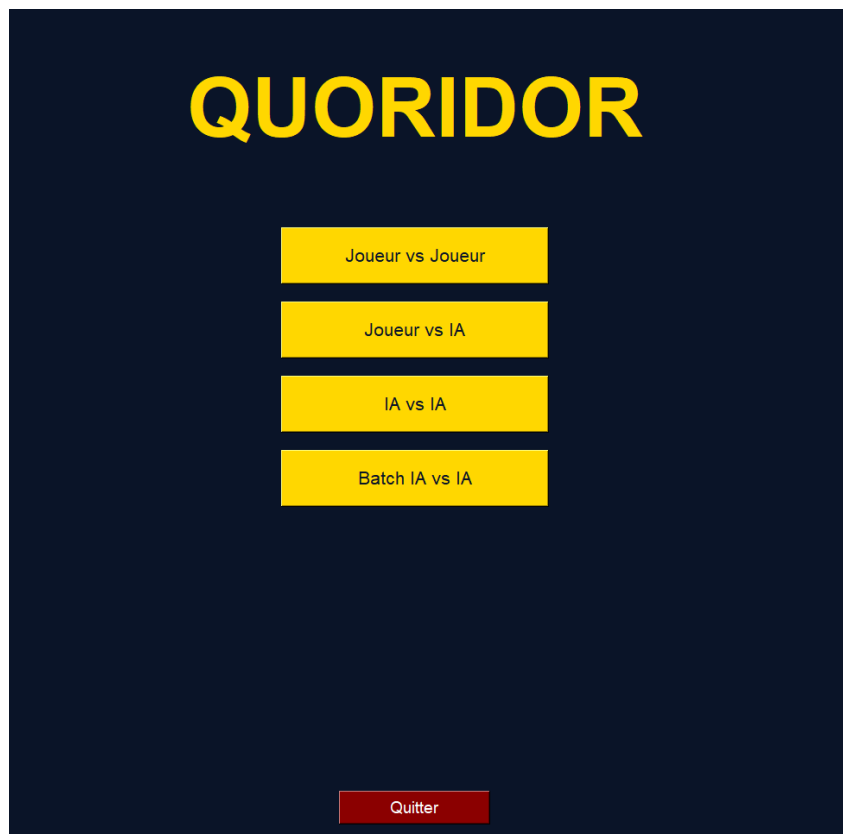


FIGURE 3 – Écran d'accueil de jeu Quorridor

L'utilisateur peut alors choisir l'un des quatre modes de jeu :

- **Joueur Vs Joueur** : deux humains s'affrontent en local.
- **Joueur Vs IA** : un joueur humain contre l'IA. Le niveau de difficulté est ensuite sélectionnable.
- **IA Vs IA** : deux IA s'affrontent, utile pour observer leurs stratégies respectives.
- **Batch IA vs IA** : lancement automatisé d'une série de parties IA vs IA pour statistiques.

4.2.2 Écran de plateau de jeu et interactions

Après avoir choisi un mode et (éventuellement) un niveau de difficulté, la fenêtre passe au plateau 9×9 :

Éléments visibles :

- Les cases claires représentent les positions possibles pour les pions.

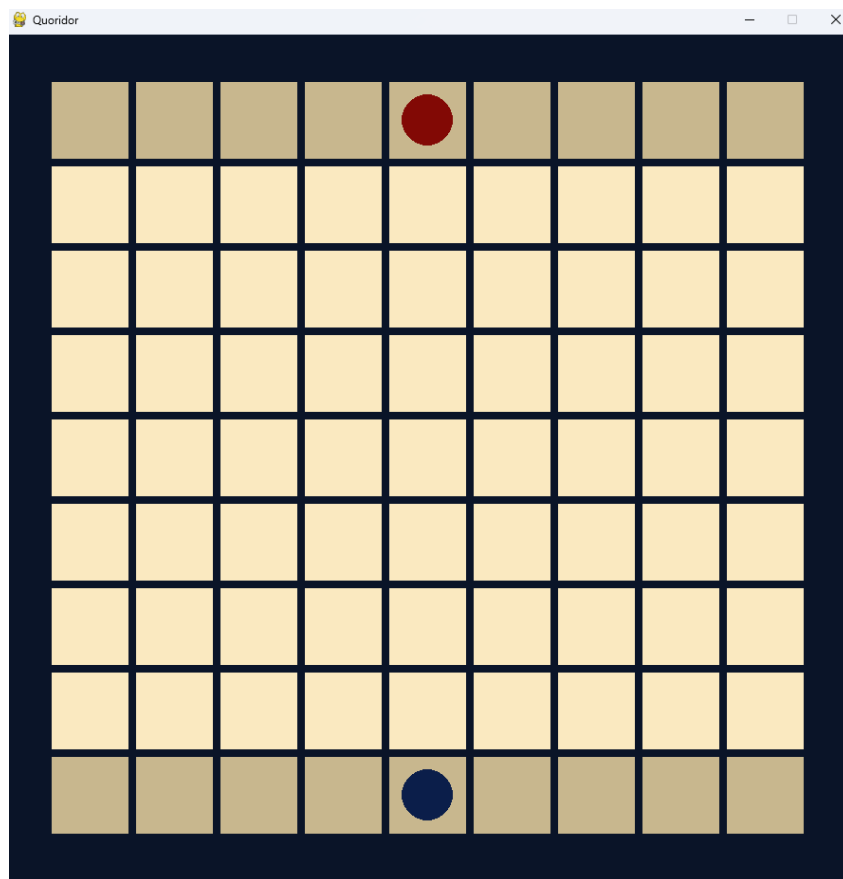


FIGURE 4 – Écran de plateau de jeu

- Les cases plus foncées en bordure indiquent les lignes de départ et d'arrivée des deux joueurs.
- Les pions sont des cercles colorés : rouge (Joueur 1) en haut, bleu (Joueur 2 ou IA) en bas.
- Le nombre de murs restants est affiché en haut à droite.

Clic sur un pion Sélectionne le pion ; les déplacements légaux apparaissent en surbrillance.

Clic sur une case vide Déplace le pion sélectionné vers cette case, si le mouvement est valide.

Clic proche d'une séparation entre cases Pose un mur (vertical ou horizontal) à l'emplacement survolé, si cela ne bloque pas l'adversaire.

Survol (hover) Une prévisualisation du mur apparaît en bleu clair tant que le curseur reste dans une zone valide.

4.3 Choix du niveau de l'IA

4.3.1 Mode « Joueur vs IA »

Après avoir choisi *Joueur vs IA*, on accède à l'écran de sélection de difficulté ([figure 5](#)), où l'on peut choisir entre *Facile*, *Intermédiaire* et *Difficile*. Selon le niveau sélectionné, l'IA utilisera des heuristiques plus ou moins sophistiquées.

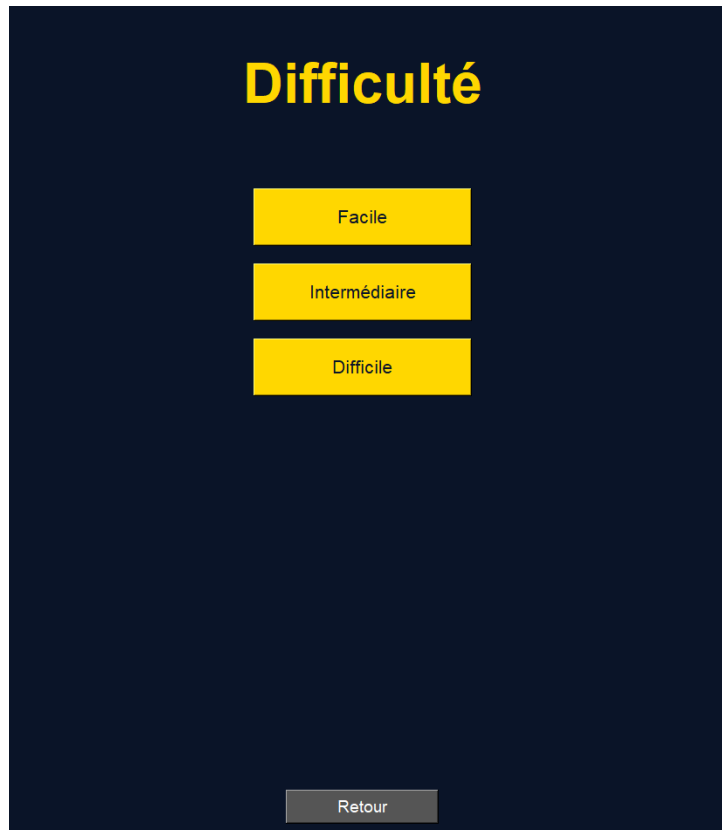


FIGURE 5 – Sélection de la difficulté en mode Joueur vs IA.

4.3.2 Mode « IA vs IA »

En mode *IA vs IA*, on sélectionne d'abord la difficulté de la première IA (rouge), puis celle de la seconde (bleu), avant de lancer la partie (figure 6). La partie se déroule ensuite automatiquement, et à la fin l'écran de victoire affiche quel camp a gagné (figure 7).

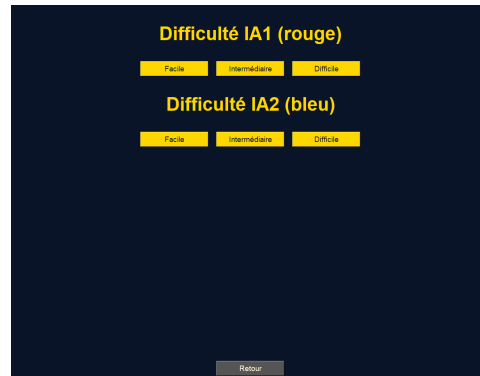


FIGURE 6 – Choix des difficultés pour IA vs IA.



FIGURE 7 – Écran de fin de partie, annonce du vainqueur et options « Menu Principal » / « Rejouer ».

4.3.3 Mode « Batch IA vs IA »

Le mode batch permet d'exécuter par défaut 100 parties consécutives entre deux IA (figure 8). On peut ajuster le nombre de matchs à l'aide des boutons « + » et « - », puis cliquer sur « LANCER ». Les résultats s'affichent directement dans la console, avec le pourcentage de victoires de chacune (figure 9).

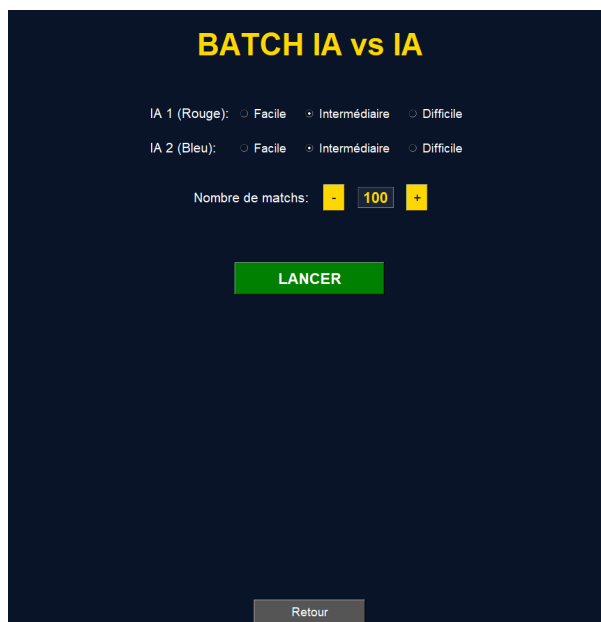


FIGURE 8 – Configuration du nombre de matchs en mode Batch IA vs IA

```
- Match 82 : Difficile gagne
- Match 83 : Difficile gagne
- Match 84 : Difficile gagne
- Match 85 : Facile gagne
- Match 86 : Difficile gagne
- Match 87 : Difficile gagne
- Match 88 : Difficile gagne
- Match 89 : Difficile gagne
- Match 90 : Difficile gagne
- Match 91 : Difficile gagne
- Match 92 : Difficile gagne
- Match 93 : Difficile gagne
- Match 94 : Difficile gagne
- Match 95 : Difficile gagne
- Match 96 : Difficile gagne
- Match 97 : Difficile gagne
- Match 98 : Facile gagne
- Match 99 : Difficile gagne
- Match 100 : Difficile gagne

Résultats finaux:
- Facile: 15 victoires (15.0%)
- Difficile: 85 victoires (85.0%)
- Matchs nuls: 0 (0.0%)
-----
```

FIGURE 9 – Exemple de sortie console après 100 parties en mode batch



FIGURE 10 – Exemple de sortie UI après 100 parties en mode batch

5 Explication des principales classes et fonctions

5.1 Classe QuoridorMenuApp (dans quoridor_menus.py)

- **Rôle** : gère l'interface Tkinter pour sélectionner mode, difficultés et nombre de matchs.
- `__init__` : initialise la fenêtre (`root`), définit le style des widgets, crée le menu principal, et prépare les constantes de difficulté.
- `_maximize_window()` : astuce cross-platform pour maximiser la fenêtre (Win/-Mac/Linux).
- `create_main_menu()` : construit la vue principale avec 4 boutons — PVP, PVE, IA vs IA, Batch IA vs IA — plus Quitter.
- `show_difficulty_menu()`, `show_ai_vs_ai_menu()`, `show_batch_ai_menu()` : affichent respectivement les sous-menus pour choisir la difficulté PVE, les deux IA, puis configurer un lot de simulations.
- `validate_numeric_input(%P)` : callback de validation pour s'assurer que l'entrée n'est que numérique et positive.
- `increment_matches()` / `decrement_matches()` : ajustent la variable `self.num_matches` via l'UI.
- `start_batch_simulation(ia1, ia2)` : lit `self.num_matches` et lance le mode 'BatchAI'.
- `start_game(mode)` : assemble le dictionnaire `game_params` (mode, difficultés, nombre de matchs) puis quitte la boucle Tkinter (`root.quit()`), pour passer au module Pygame.

5.2 Exception ReturnToMenu (dans Projet IA.py)

- Hérite de `Exception`.
- Utilisée pour interrompre proprement une boucle de jeu Pygame et remonter au menu principal.

5.3 Fonction `launch_game(game_params)`

- Point d'entrée du module Pygame.
- Si `game_params` est `None` : appelle `main_menu()` (menu Pygame).
- Sinon extrait `mode` et délègue à :
 - `mainPVP()` pour 'PVP'
 - `mainPVE(difficulty)` pour 'PVE'
 - `mainAIvsAI(d1,d2)` pour 'AIvsAI'
 - `run_batch_simulations(d1,d2,n)` puis `main_menu()` pour 'BatchAI'

- Gère le chargement des polices, la création de la fenêtre Pygame et le reset des variables globales.

5.4 Boucles principales de jeu

`mainPVP()` Tour à tour humain vs humain :

- Gère les clics souris pour déplacer ou poser un mur.
- Met à jour l’affichage et détecte la victoire.

`mainPVE(difficulte)` Humain vs IA :

- Idem PVP pour le joueur, puis
- IA calcule `meilleur_coup_ia(...)` et l’exécute (déplacement ou mur).

`mainAIvsAI(d1,d2)` Simulation graphique IA vs IA ; et

`simulate_ai_vs_ai(d1,d2)` version sans UI pour batch.

5.5 IA et évaluation

- `a_star_search()` : recherche de chemin A* (heuristique Manhattan).
- `evaluer_position()`, `evaluer_position_intermediaire()`, `evaluer_position_difficile()` : trois fonctions d’évaluation selon le niveau.
- `minimax(..., difficile)` : implémentation unifiée de Minimax avec alpha–beta, ajuste la profondeur et la sélection de coups en fonction de la difficulté.
- `meilleur_coup_ia(...)` : wrapper qui choisit entre déplacement (`meilleur_deplacement_pour_joueur`) et placement de mur (`meilleur_mur_pour_joueur`), selon une probabilité modulée par la situation de jeu.

6 Analyse des heuristiques dans les trois fonctions d'évaluation

Voici une explication détaillée des heuristiques utilisées dans les trois fonctions d'évaluation pour les différents niveaux de difficulté de l'IA dans le jeu Quoridor.

6.1 IA Facile (`evaluer_position`)

Cette fonction implémente une heuristique simple avec des coefficients réduits pour rendre l'IA moins efficace.

Heuristiques principales

- *Distance au but* : favorise les positions où le joueur est proche de son objectif et l'adversaire loin du sien.
- *Progression vers l'objectif* : bonus pour l'avancement sur le plateau en direction de l'objectif.
- *Position centrale* : léger bonus pour contrôler les colonnes centrales du plateau.
- *Facteur aléatoire* : important facteur aléatoire (-2.0 à 2.0) pour rendre l'IA moins prévisible.

Coefficients réduits

```
1 poids_distance = 3.0          # réduit par rapport aux autres difficulté
  s
2 poids_avance = 2.0            # également réduit
3 poids_position_centrale = 0.3 # très faible impact
```

Listing 2 – Classe et décorateurs pour notifications

6.2 IA Intermédiaire (`evaluer_position_intermediaire`)

Cette fonction implémente une heuristique plus orientée vers le blocage de l'adversaire que vers l'avancée personnelle.

Heuristiques principales

- *Distance adversaire vs joueur* : coefficients déséquilibrés pour favoriser le blocage.
- *Conservation des murs* : valorise davantage la conservation des murs pour un usage stratégique.
- *Bonus de blocage* : avantage spécial quand le joueur est proche de l'adversaire et peut potentiellement le bloquer.

- *Contrôle du centre* : forte valorisation du contrôle du centre, surtout dans la moitié adverse.
- *Pénalité d'écartement* : pénalité pour s'éloigner trop du centre horizontalement.

Coefficients remarquables

```
1 poids_distance_adversaire = 7.0 # très élevé, focalisation sur le
   blocage
2 poids_distance_joueur      = 3.0 # plus faible, moins d'importance à sa
   propre progression
3 poids_murs_restants        = 1.2 # valorise la conservation des murs
```

Listing 3 – Classe et décorateurs pour notifications

6.3 IA Difficile (evaluer_position_difficile)

Cette fonction implémente l'heuristique la plus sophistiquée avec un équilibre entre progression personnelle et blocage de l'adversaire.

Heuristiques principales

- *Distance équilibrée* : évaluation équilibrée des distances relatives au but.
- *Progression avancée* : valorise fortement l'avancement vers l'objectif.
- *Contrôle du centre nuancé* : bonus dégressif depuis la colonne centrale, avec une zone d'influence plus large.
- *Positions stratégiques* : bonus pour les positions avancées dans le territoire adverse.
- *Flexibilité de mouvement* : évaluation des chemins alternatifs disponibles, favorisant les positions avec plus d'options.
- *Utilisation stratégique des murs* : stratégie qui varie selon la phase de jeu (début, milieu, fin).

Phases de jeu distinctes

```
1 # En début de partie (moins de 8 murs posés)
2 murs_score = murs_restants_joueur * 0.4
3
4 # En milieu de partie (8 16 murs posés)
5 ratio_joueur_adverse = murs_restants_joueur / (murs_restants_adversaire
   + 0.1)
6 murs_score = ratio_joueur_adverse * poids_murs_strategie
7
8 # En fin de partie
9 if dist_joueur < dist_adversaire:
```

```
10     # en avance : bloquer l'adversaire
11     murs_score = murs_restants_joueur * 0.8
12 else:
13     # en retard : valeur moindre des murs
14     murs_score = murs_restants_joueur * 0.3
```

Listing 4 – Classe et décorateurs pour notifications

6.4 Comparaison des approches

1. **IA Facile** : approche simplifiée avec peu de planification stratégique et beaucoup d'aléatoire.
2. **IA Intermédiaire** : stratégie orientée obstruction, privilégiant le blocage de l'adversaire plutôt que l'avancée directe.
3. **IA Difficile** : stratégie équilibrée et adaptative qui évolue selon la phase de jeu et la position relative des joueurs, avec une évaluation plus fine des positions centrales et des chemins alternatifs.

Ces différentes fonctions d'évaluation permettent de créer une progression de difficulté naturelle pour le joueur humain, tout en maintenant des styles de jeu distincts et reconnaissables.

7 Évaluation expérimentale des IA : résultats finaux

Pour quantifier la performance relative des trois niveaux d'IA, nous avons lancé, pour chaque configuration, 100 parties consécutives et relevé le pourcentage de victoires des joueurs (J1 vs J2) et de matchs nuls.

Remarque

Dans les règles originales de Quoridor, la notion de « match nul » n'existe pas : la partie se termine dès qu'un joueur atteint la ligne adverse. Cependant, lors de nos simulations IA vs IA (100 parties consécutives), certaines parties peuvent durer longtemps, ce qui retarde le lancement des parties suivantes. Pour garantir la robustesse et la reproductibilité, nous avons donc fixé une limite de 200 tours par partie : toute partie dépassant ce seuil est automatiquement considérée comme nulle. Sur les 900 parties simulées (9 configurations \times 100 parties), 5 ont été classées comme nulles, soit 0,55 %.

7.1 Protocole de test

- **Auto-affrontements** (même niveau contre lui-même) pour mesurer l'avantage du joueur 1 (premier coup).
- **Affrontements croisés** : chaque niveau affronte les deux autres dans les deux sens (J1 vs J2 et J2 vs J1).

7.2 Analyse du nombre de victoires IA vs IA

7.2.1 Tableau de synthèse des résultats

Match	Victoires J1 (%)	Victoires J2 (%)	Nuls (%)
Facile vs Facile	53	47	0
Intermédiaire vs Intermédiaire	67	33	0
Difficile vs Difficile	59	38	3
Facile vs Intermédiaire	32	68	0
Intermédiaire vs Facile	82	18	0
Facile vs Difficile	7	93	0
Difficile vs Facile	91	9	0
Intermédiaire vs Difficile	35	63	2
Difficile vs Intermédiaire	82	18	0

TABLE 1 – Synthèse des pourcentages de victoires sur 100 parties par configuration d'IA.

7.2.2 Analyse des données

1. **Avantage du premier joueur** Sur les auto-affrontements, J1 remporte systématiquement plus de parties :

- Facile vs Facile : 53 % vs 47 %
- Intermédiaire vs Intermédiaire : 67 % vs 33 %
- Difficile vs Difficile : 59 % vs 38 %, 3 % de nuls

Cet avantage est maximal pour le niveau intermédiaire (différence de 34 points).

2. **Différences de puissance entre niveaux**

- *Intermédiaire vs Facile* : 82 % de victoires pour Intermédiaire (J1) contre 18 % pour Facile.
- *Difficile vs Facile* : 91 % vs 9 %, écart de 82 points.
- *Difficile vs Intermédiaire* : 82 % vs 18 %, écart de 64 points.

La hiérarchie (Difficile > Intermédiaire > Facile) est très nette, avec un écart croissant à chaque palier.

3. **Asymétries selon l'ordre de jeu** Inverser les rôles (J2 vs J1) confirme : lorsque le plus fort commence, son taux de victoire dépasse souvent 90 % (par ex. Difficile vs Facile : 93 % pour J2).

4. **Implications**

- Le premier coup confère un avantage non négligeable, même sur des IA identiques.
- Les heuristiques et la profondeur de recherche jouent un rôle décisif : chaque niveau supérieur écrase largement le précédent.
- En tournoi, pour équilibrer les chances, il serait judicieux d'alterner systématiquement l'ordre de jeu.

7.3 Analyse du nombre moyen de coups par partie IA vs IA

L'objectif de cette partie est de déterminer, pour chaque configuration d'IA, le nombre moyen de coups nécessaires à l'une des deux intelligences artificielles pour atteindre la victoire. Cette métrique nous permet d'évaluer si l'IA parvient à optimiser son jeu en minimisant la longueur des parties et, par conséquent, sa performance stratégique.

7.3.1 Tableau de synthèse

Match	nombre moyen de coups sur 100 matchs
Facile vs Facile	29.8
Intermédiaire vs Intermédiaire	21.1
Difficile vs Difficile	22.4
Facile vs Intermédiaire	26.1
Intermédiaire vs Facile	25.5
Facile vs Difficile	22.1
Difficile vs Facile	21.4
Intermédiaire vs Difficile	23.4
Difficile vs Intermédiaire	20.6

TABLE 2 – Synthèse des pourcentages de victoires sur 100 parties par configuration d'IA.

7.3.2 Interprétabilité du tableau de synthèse

Voici quelques points clés pour interpréter les résultats :

1. Parties « équilibrées » (même niveau vs même niveau)

- **Facile vs Facile (29,8 coups)** Les IA « faciles » progressent lentement et posent peu de murs, d'où des parties plus longues.
- **Intermédiaire vs Intermédiaire (21,1 coups)** Les IA intermédiaires bloquent et avancent de façon plus structurée, raccourcissant nettement la durée.
- **Difficile vs Difficile (22,4 coups)** Les IA « difficiles » sont plus stratégiques mais parfois trop prudentes, ce qui allonge légèrement par rapport au niveau intermédiaire.

2. Parties « déséquilibrées » (différents niveaux)

- *Lorsque l'IA la plus forte commence*
 - Difficile vs Intermédiaire : 20,6 coups
 - Difficile vs Facile : 21,4 coups

L'IA « difficile » exploite son avantage tactique pour conclure rapidement.
- *Lorsque l'IA la plus faible commence*
 - Intermédiaire vs Difficile : 23,4 coups
 - Facile vs Difficile : 22,1 coups

Le premier coup de l'IA plus faible allonge un peu la partie, mais l'adversaire « difficile » domine toujours.

3. Impact de l'ordre de jeu

- L'IA qui joue en premier réduit légèrement la durée de la partie (ex. 26,1 vs 25,5 coups pour Facile vs Intermédiaire), mais cet effet reste faible comparé à l'écart de niveau.

Résumé :

- Plus l'écart de compétence est grand, plus la partie se termine en peu de coups.
- Les IA faciles génèrent les affrontements les plus longs (>29 coups).
- Les IA intermédiaires produisent les matches les plus rapides (21 coups).
- Les IA difficiles restent proches du niveau intermédiaire (21–22 coups).
- L'avantage du premier coup diminue modestement le nombre moyen de tours.

8 Bilan du projet

Le rapport est structuré en trois volets principaux :

1. Conception et architecture

- Contexte et objectifs pédagogiques : implémentation d'algorithmes d'IA et interface graphique.
- Choix de Quoridor : jeu à information parfaite, complexité modulable.
- Organisation modulaire : script de lancement, menus Tkinter, moteur Pygame + IA.
- Bibliothèques et paradigmes : Python 3, Tkinter, Pygame, A, Minimax -, heuristiques selon niveau.

2. Implémentation et logique de jeu

- Classe QuoridorMenuApp pour les menus et script Projet IA.py pour la logique du plateau.
- Trois fonctions d'évaluation (`evaluer_position`, `evaluer_position_intermediaire`, `evaluer_position_difficile`) offrant des heuristiques graduées : aléatoire, blocage, stratégie équilibrée.
- Algorithmes : A pour la recherche de chemin, Minimax avec élagage -, profondeur et aléa ajustés selon la difficulté.

3. Évaluation expérimentale

- *Résultats de victoire* : hiérarchie claire Difficile > Intermédiaire > Facile, avantage net du premier joueur.
- *Gestion des matchs nuls* : limite à 200 tours (5 nuls sur 900, soit 0,55 %).
- *Durée de partie (nombre moyen de coups)* :
 - IA Facile : $\approx 29,8$ coups
 - IA Intermédiaire : $\approx 21,1$ coups
 - IA Difficile : $\approx 22,4$ coups
 - Confrontations déséquilibrées : moins de 21 coups en moyenne pour l'IA la plus forte.

Points forts du projet

- Séparation claire des responsabilités : interface, moteur de jeu, IA.
- Modularité des fonctions d'évaluation et paramétrage aisé.
- Robustesse expérimentale : simulations batch avec gestion des cas extrêmes.

Perspectives d'amélioration

- Optimisation des performances (parallélisation, mise en cache).

- Enrichissement de l'interface (affichage des temps de jeu, visualisation des parcours A).
- Extension du jeu : mode multijoueur réseau, variantes de plateau, IA supplémentaires.

Références

- [1] ELISE BONZON. *Diapositives du cours d'Intelligence Artificielle*. Support de cours en ligne, UFR math-info, Université Paris cité. 2025.
- [2] GEEKSFORGEEKS. *A* Search Algorithm*. GeeksforGeeks. Consulté le 1er mai 2025. 2024. URL : <https://www.geeksforgeeks.org/a-search-algorithm/>.
- [3] GEEKSFORGEEKS. *Alpha-Beta Pruning in Minimax Algorithm*. GeeksforGeeks. URL : <https://www.geeksforgeeks.org/alpha-beta-pruning-in-game-theory/>.
- [4] GEEKSFORGEEKS. *Breadth First Search or BFS for a Graph*. GeeksforGeeks. URL : <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>.
- [5] GEEKSFORGEEKS. *Minimax Algorithm in Game Theory*. GeeksforGeeks. URL : <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-and-artificial-intelligence/>.
- [6] GIGAMIC. *Quoridor – Règles officielles*. Site officiel. URL : <https://www.gigamic.com/quoridor>.
- [7] *Quoridor*. Wikipedia, l'encyclopédie libre. URL : <https://fr.wikipedia.org/wiki/Quoridor>.