

Rapport de Projet

POLYQUARTO – ALGO & COMPLEXITÉ

ARMANET Nathan & NAAJI Dorian – 3A INFO GROUPE 2-H

POLYTECH LYON

ARMANET.NATHAN@ETU.UNIV-LYON1.FR

DORIAN.NAAJI@ETU.UNIV-LYON1.FR

Table des matières

1. Introduction	3
1.1. Le jeu du Quarto	3
1.2. Présentation du programme	3
1.2.1. Côté Utilisateur.....	3
1.2.2. Côté Développeur.....	6
2. Choix technologiques	8
2.1. Langage, compilateur & environnement de développement	8
2.2. Librairie graphique.....	8
2.3. GIT.....	8
3. Organisation	9
3.1. Gestion de projet.....	9
3.2. Répartition des tâches.....	9
4. Algorithmes utilisés	9
4.1. IA du Quarto	9
4.2. Conditions de victoire.....	10
5. Documentation d'utilisation	10
6. Instructions de compilation	10
7. Bonus : Tic-Tac-Toe	10
7.1. Présentation sommaire	10
7.2. Algorithmes utilisés	10

1. Introduction

1.1. Le jeu du Quarto

Le jeu du Quarto est un jeu à deux joueurs consistant à aligner quatre pièces ayant au moins un point commun entre elles. La particularité du jeu réside dans le fait que les pièces que l'un des deux joueurs placera est choisie par son adversaire.

Le jeu est constitué d'un tablier de seize cases (4x4) et de seize pions différenciables par 4 caractéristiques :

- Leur taille : grand ou petit
- Leur couleur : rouge ou bleu, ou autre selon design du jeu
- Leur sommet : plein ou troué
- Leur forme : Rectangulaire ou cylindrique.

1.2. Présentation du programme

1.2.1. Côté Utilisateur

En lançant le programme, l'utilisateur arrive tout d'abord sur une page d'accueil, où il peut choisir son mode de jeu : Contre l'IA ou contre un autre joueur.

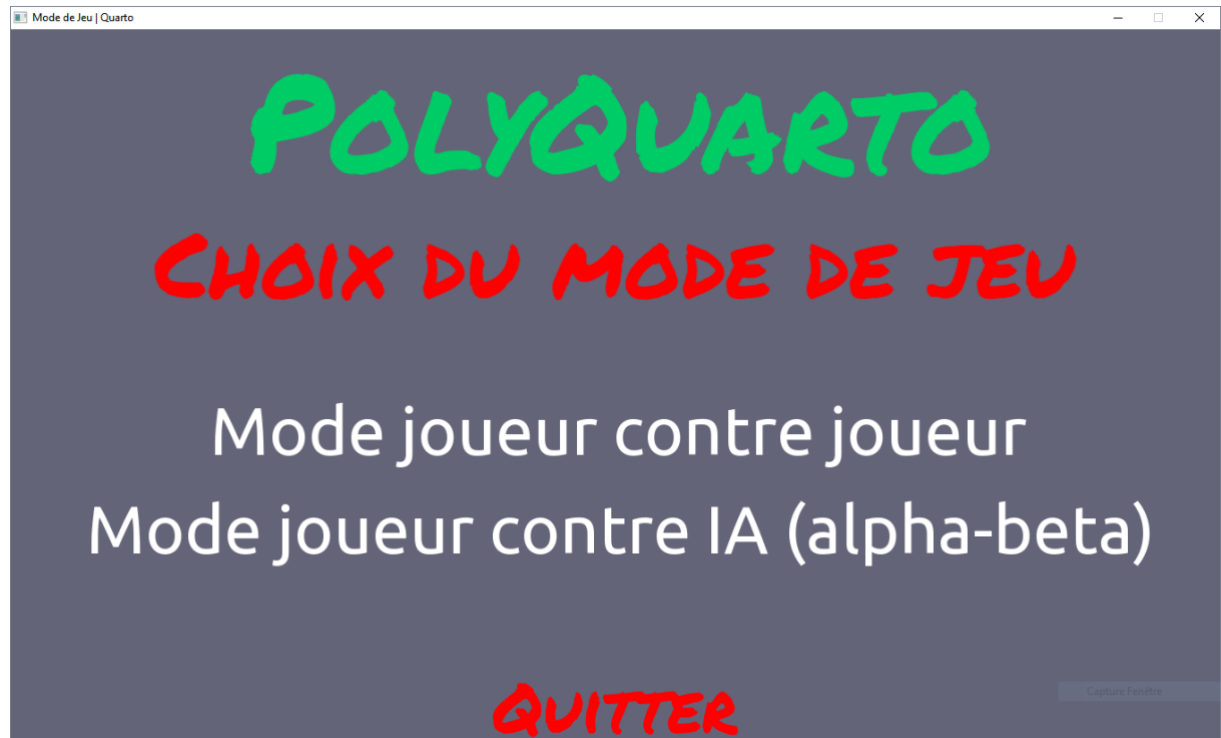


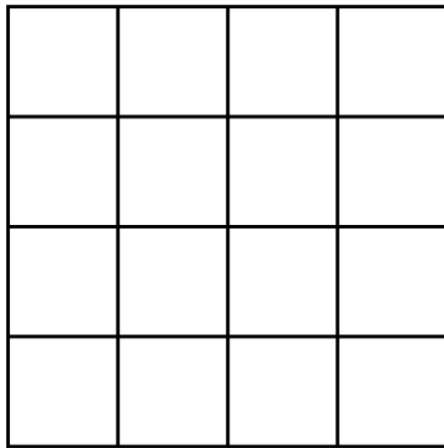
FIGURE 1 : ACCUEIL DU PROGRAMME

En choisissant le jeu contre l'IA, le joueur tombe directement sur l'écran de jeu et commence une partie contre l'ordinateur. En revanche, en choisissant une partie contre un joueur, l'utilisateur arrive sur l'écran ci-dessous




FIGURE 2 : CHOIX DES REGLES EN JOUEUR CONTRE JOUEUR

Ce dernier peut alors choisir de jouer en mode classique, en alignant 4 pions ayant les mêmes caractéristiques horizontalement, verticalement ou diagonalement. Il peut sinon choisir de jouer en « Mode Tetris ». En effet, les joueurs se mettent d'accord sur un motif à réaliser pour la victoire. Une fois le motif choisi, la partie commence et le but est alors d'aligner 4 pions de caractéristiques similaires mais en respectant le motif choisi pour gagner. Le motif choisi est également rappelé dans l'écran de jeu.



J2 choisit le pion et
J1 le place.

Motif choisi : 

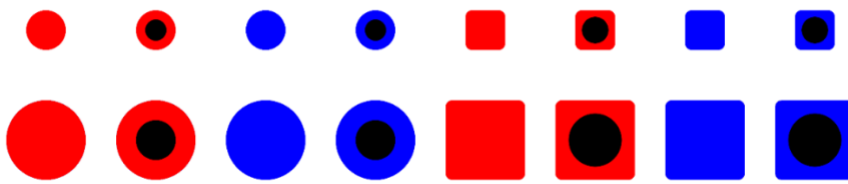
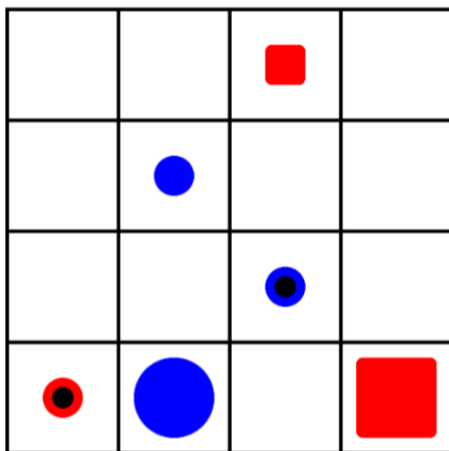


FIGURE 3 : JOUEUR CONTRE JOUEUR AVEC MOTIF TETRIS

Les joueurs peuvent alors s'affronter.



J2 choisit le pion et
J1 le place.

Motif choisi : 

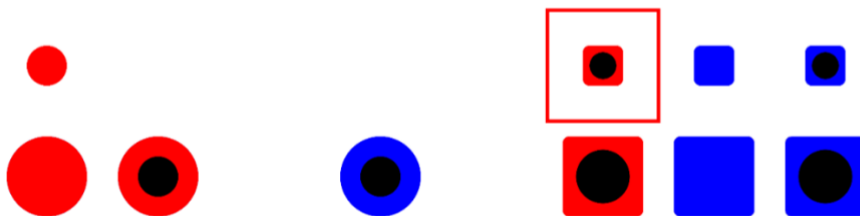


FIGURE 4 : AFFRONTMENT DES DEUX JOUEURS

Une fois la partie terminée (victoire d'un des joueurs ou match nul), l'utilisateur a alors la possibilité de retourner aux modes de jeu ou de quitter le programme.



FIGURE 5 : ÉCRAN DE FIN DE PARTIE.

1.2.2. Côté Développeur

Le programme est composé de 6 classes :

- Case : Une case de la grille, elle contient un pion
- Grille : La grille de jeu, elle contient l'ensemble des cases
- IA : Gère l'IA
- Jeu : Gère l'affichage & les boucles de jeu.
- Motif : Une énumération permettant de gérer les motifs du Tetris
- Pion : Un pion d'une case et ses différentes caractéristiques.

Ci-après un diagramme de classes représentant l'architecture globale du programme.

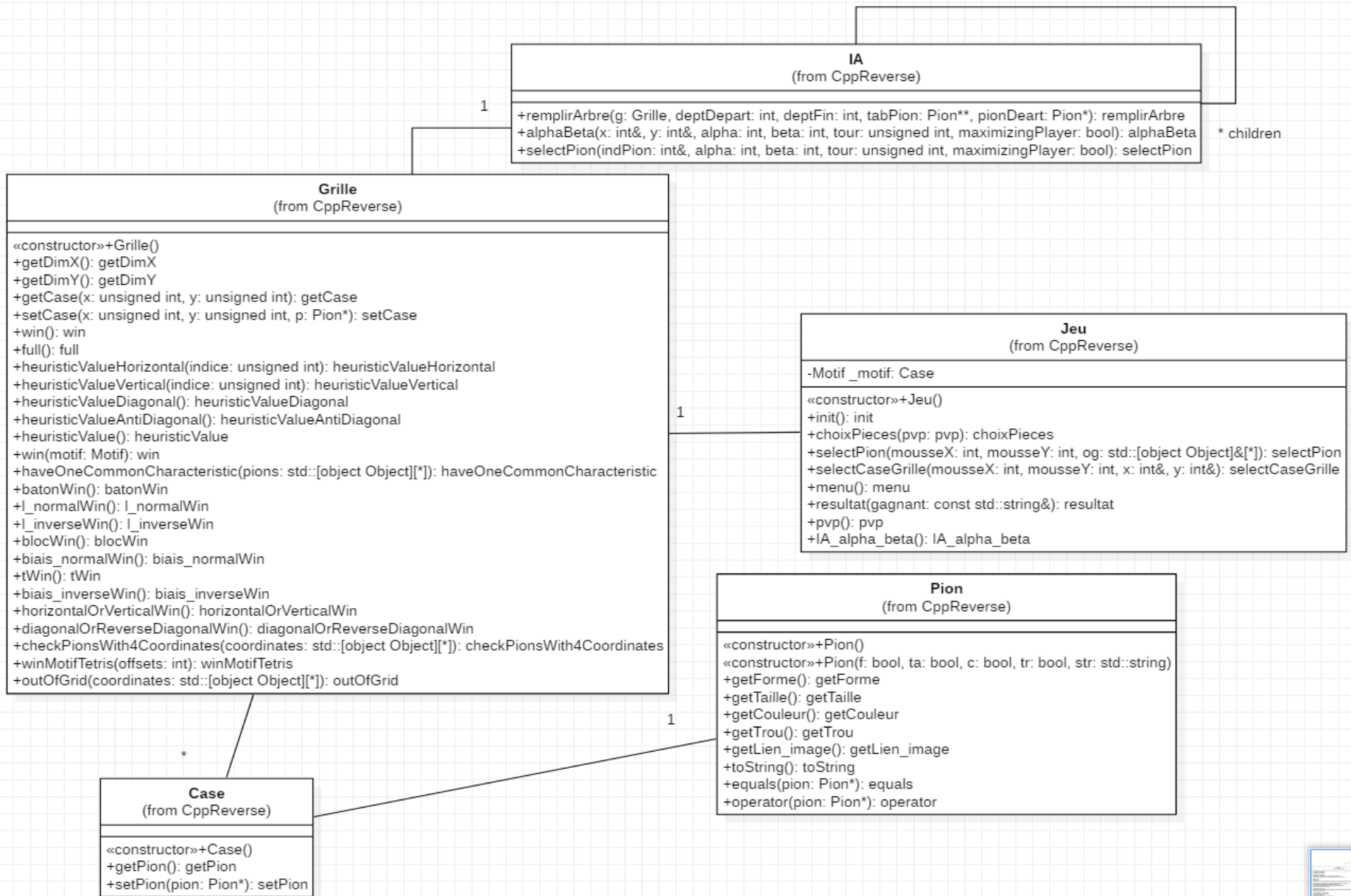


FIGURE 6 : DIAGRAMME DE CLASSES

2. Choix technologiques

2.1. Langage, compilateur & environnement de développement

Le langage utilisé pour développer l'application était le C++. Nous avons utilisé l'outil [CLion](#) de la suite [Jetbrains](#), un environnement de développement pratique et ergonomique. Nous avons développé en binôme et utilisons pour l'un un ordinateur sous Windows et pour l'autre un ordinateur sous Mac OS. Nous avons utilisé des compilateurs respectivement compatibles pour nos systèmes d'exploitation, notamment MinGW pour Windows.

2.2. Librairie graphique

Pour la réalisation d'un tel projet, notre choix s'est orienté vers l'utilisation de la bibliothèque graphique [SFML](#) : Simple and Fast Multimedia Library. D'où son nom, la bibliothèque est simple à prendre en main et permet de rapidement réaliser des interfaces graphiques en C++, grâce à une communauté largement présente, via des forums de discussions et même un serveur Discord. SFML propose également un panel de tutoriels efficaces. Sa documentation est également très claire, la librairie est de plus documentée aussi bien en anglais qu'en français.

2.3. GIT

Pour versionner, organiser et partager la réalisation du code, nous avons opté pour l'outil GIT, via un dépôt GitHub. (<https://github.com/NathanARMANET/quarto>). Cet outil nous a permis de rapidement être productif en binôme, de se partager les tâches mais plus généralement de travailler en simultané sur le même projet.

3. Organisation

3.1. Gestion de projet

La gestion de ce projet s'est basée sur une méthodologie agile, avec un fonctionnement en cycles. À chaque début de cycle, un ensemble de tâches était défini, ce dernier devait ensuite être terminé avant le début du prochain cycle. L'ajout de code se faisait ainsi fonctionnalité par fonctionnalité. Nous avons également utilisé des formalismes simples (to do lists) pour une répartition efficace des tâches.

3.2. Répartition des tâches

Nathan ARMANET a principalement développé les bases fonctionnelles du programme, en mettant en place le système de Grille et de Cases, tant bien au niveau métier qu'au niveau graphique

Dorian NAAJI a eu pour tâche de gérer les conditions de victoire, la gestion des motifs du jeu TETRIS et un menu de choix des règles.

Les tâches ont été réparties sur la base de l'initiative.

4. Algorithmes utilisés

4.1. IA du Quarto

Nous avons utilisé l'algorithme alpha-beta qui permet à l'ordinateur de choisir le meilleur coup selon l'état actuelle de la grille. Cet algorithme est coûteux, en effet à un tour i compris entre 1 et 16, l'IA doit créer un arbre qui à $((16 - i + 1)!)^2 = ((17 - i)!)^2$, ce nombre s'explique par le fait que toutes les pièces jouables sont 2 à 2 différentes : nous devons donc créer une situation pour chaque couple pièce/case. C'est pour cela que nous ne créons qu'une partie de l'arbre et que nous calculons une valeur heuristique pour les situations présente dans l'arbre.

Concernant l'IA optimisé, nous n'avons malheureusement pas réussi à l'implémenter.

4.2. Conditions de victoire

Les conditions de victoires sont gérées par des algorithmes simples, parcourant la grille (boucles imbriquées) puis vérifient qu'un motif donné est réalisé en fonction des coordonnées relatives au variables de boucle.

5. Documentation d'utilisation

Le jeu est démarrable grâce au fichier exécutable « quarto.exe », dans le dossier bin/ à la racine du projet. Il faut pour cela le lancer depuis un invite de commande, en veillant à avoir MinGW installé si l'utilisateur est sur un système Windows.

6. Instructions de compilation

Pour la compilation du projet, Nous avons créé un MakeFile permettant de compiler correctement l'exécutable de projet ainsi que de nettoyer l'archive des fichiers objets (.o) et aussi de l'exécutable.

Sous Windows, le programme est ainsi compilable grâce à la commande « mingw32-make » dans le dossier du projet. Il faut bien sûr veiller à avoir installé MinGW auparavant, et à l'avoir ajouté aux variables d'environnement de Windows.

7. Bonus : Tic-Tac-Toe

7.1. Présentation sommaire

Le jeu du morpion est un jeu au tour par tour entre 2 adversaires (Joueur contre Joueur ou Joueur contre IA). Les adversaires jouent sur une grille de 9 cases (3x3) et doivent essayer d'aligner 3 symboles identiques (soit des « X », soit des « O ») de manière horizontale, vertical ou diagonale. Ces symboles sont définis avant le début de la partie (ex : les « X » pour le joueur 1 et les « O » pour le joueur 2/IA).

7.2. Algorithmes utilisés

Nous avons implémenté 2 algorithmes.

Le premier est l'algorithme codé est l'algorithme min-max qui permet à l'ordinateur de choisir le meilleur coup selon l'état actuelle de la grille. Cet algorithme est coûteux, en effet à un tour i compris entre 1 et 9, l'IA doit créer un arbre qui à $((9 - i + 1)!) = ((10 - i)!)$.

Le second algorithme implémenter est une optimisation du premier algorithme. Elle est nommé alpha-beta et permet de « sauter » des situations lorsque l'on sait qu'il n'y aura aucune chance que l'ordinateur la considère (une meilleure situation, ou une pire, c'est déjà présenté).