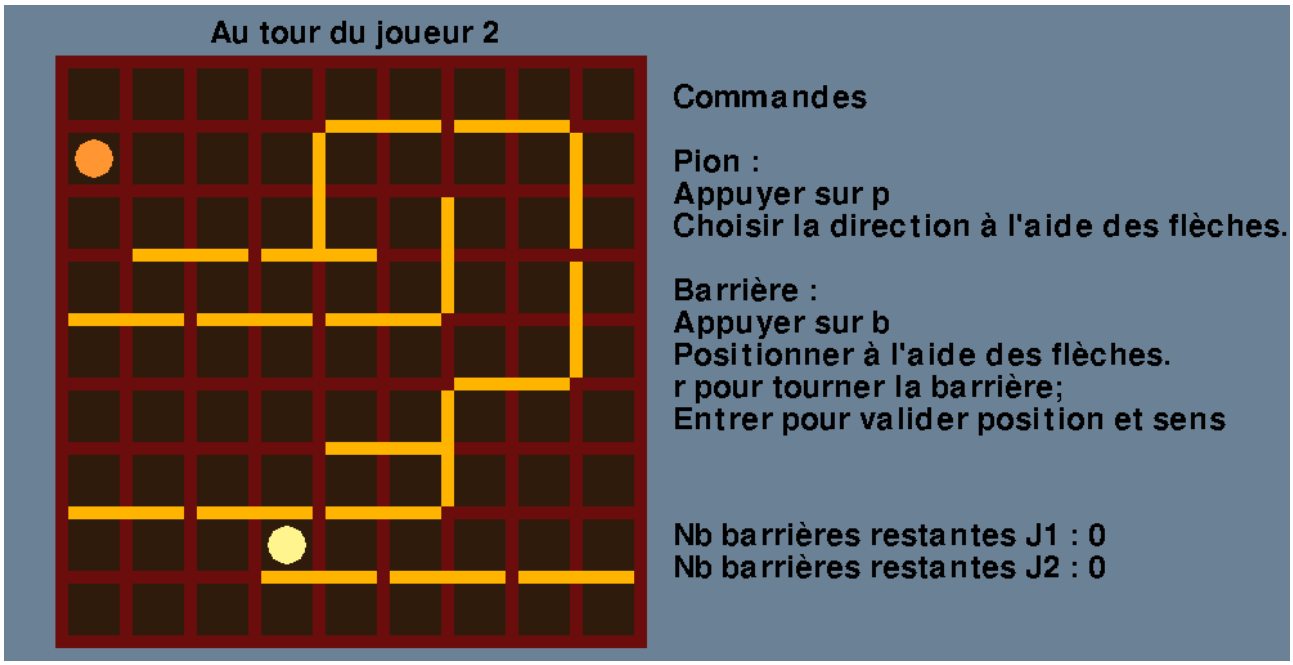
RAVOT Bastien TS4

Spécialité Informatique et Sciences du Numérique
Dossier de présentation de projet :

Un jeu de stratégie : Le Quoridor



Projet réalisé avec Anthony BACCUET lors de l'année scolaire 2014-2015

SOMMAIRE

I-Présentation du projet 1

A-Le jeu de plateau de Gigamic : Quoridor 1

B-Les objectifs initiaux du projet 1

C-La répartition des tâches 2

D-L'environnement de travail 2

II-Présentation générale de l'algorithme 3

A-Représenter le plateau 3

B-La structure du programme 3

C-Le fonctionnement de la classe Jouer 4

D-Le principe de fonctionnement de l'algorithme A* 5

III-Explication détaillée de certaines fonctionnalités 7

A-La récupération des entrées clavier 7

B-Autoriser la pose d'une barrière 7

Conclusion 8

Annexe : le code python complet 9

I-Présentation du projet

A-Le jeu de plateau de Gigamic : Quoridor

Quoridor est un jeu de stratégie inventé par Mirko Marchesi et édité par Gigamic. Il se joue à 2 ou 4 joueurs. Chaque joueur possède son propre pion qui évolue sur un plateau carré, de 9 cases de côté. Au début de la partie, les pions sont placés chaque au milieu d'un bord différent du plateau (de manière à se faire face s'il n'y a que 2 joueurs).

Le but de chaque joueur est d'atteindre le bord opposé en premier (il peut atteindre n'importe laquelle des 9 cases composant ce bord pour l'emporter).

Avant de commencer, 20 barrières sont équitablement réparties entre les joueurs (10 chacun s'il sont 2, 5 s'ils sont 4). Ces barrières font deux cases de long et seront placées dans des espaces entre les cases, de manière à ne pas empiéter sur une troisième case. Une fois posées elles ne pourront plus être déplacées ni retirées et devront être contournées par tout les joueurs.

Les joueurs jouent les uns après les autres, dans un ordre immuable. On ne peut pas passer son tour et on peut forcément faire quelque chose. A son tour, le joueur possède deux options : avancer son pion ou poser une barrière, s'il lui en reste.

S'il choisit d'avancer son pion, il devra se déplacer sur une case adjacente, mais pas en diagonale. Il se déplace d'une seule case par tour et ne peut ni traverser de barrière ni sortir du plateau.

Si le pion d'un adversaire se trouve sur une case atteignable par le joueur, ce dernier peut décider de lui sauter par-dessus car deux pions ne peuvent pas se trouver simultanément sur la même case. Dans ce cas, il devra garder la même direction si possible (et donc finalement avancer de deux cases en ligne droite), sinon il choisira une seconde direction pour effectuer son saut (au final, cela reviendra à un déplacement en diagonale).

S'il préfère poser une barrière, il devra s'assurer que cette dernière n'empêche pas un pion d'atteindre son bord d'arrivée.

Finalement, Quoridor est donc un jeu de labyrinthe où le hasard n'influe pas sur le résultat de la partie. Le mode 4 joueurs peut amener des stratégies d'alliance.

Il a été récompensé par de nombreux prix lors de sa sortie en 1997, comme celui le Grand Prix du Jouet en France, et le Prix d'Excellence des Consommateurs au Canada. Il apparaît également dans le top 5 des meilleurs jeux selon Mensa, aux Etats-Unis. Il a aussi été récompensé en Allemagne.

B-Les objectifs initiaux du projet

Nous souhaitions commencer par programmer le jeu de manière à ce que 2 joueurs puissent s'affronter sur la même machine, qui veillerait au respect des règles, le tout avec une interface minimale, textuelle.

Nous envisagions ensuite d'améliorer l'interface, en restant en 2D, ou en nous lançant dans le 3D via Blender.

Ensuite, une intelligence artificielle (IA) pourrait être programmée afin de permettre à un utilisateur de jouer seul. Au départ, cet IA pourrait jouer de manière totalement aléatoire, puis ensuite elle pourrait être améliorer pour chercher le meilleur coup possible, éventuellement en prévoyant à plusieurs coups.

Par la suite, l'ajout d'un mode 4 joueurs, dont 0 à 3 IA paraissait envisageable, tout comme l'ajout d'une page web pour enregistrer et afficher les résultats des joueurs voire même leur permettre de jouer en réseau.

C-La répartition des tâches

Nous avons partagé le travail concernant les fonctions permettant à deux joueurs de s'affronter dans le respect des règles.

En outre, Anthony c'est chargé de l'IA alors que je me suis occupé de l'algorithme de recherche du chemin le plus court, et de la partie concernant l'affichage.

Faute de temps suffisant, l'affichage est resté en 2D, géré par le module pygame. L'IA ne se contente de chercher le coup le plus intéressant pour elle à un moment donné, sans penser aux ripostes possibles de l'adversaire ni à ce avec quoi elle enchainera. Cependant, cela est suffisant pour battre un débutant et même des joueurs plus expérimentés environ une fois sur deux.

La page web et le mode 4 joueurs n'ont pas été faits.

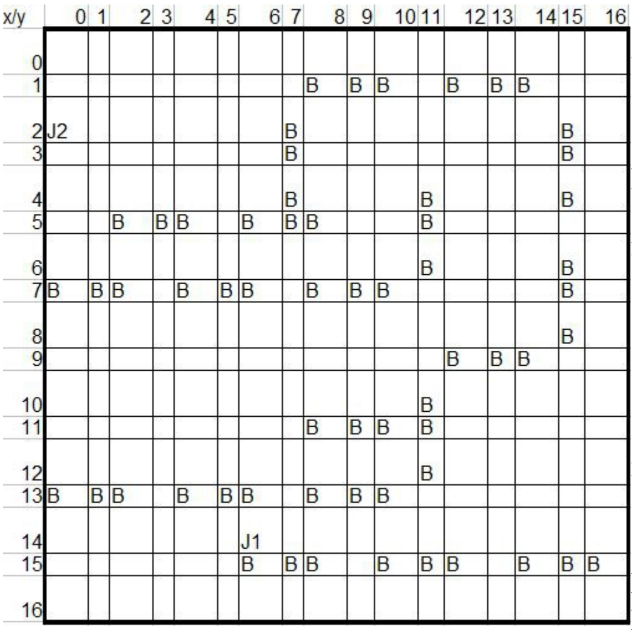
D-L'environnement de travail

Le langage, python, était imposé. Nous avons utilisé l'éditeur Geany pour écrire les scripts.

Différents modules python nous ont été utiles, à commencer par copy et plus précisement sa fonction deepcopy qui nous a permis de créer des copies de la liste représentant le plateau et de modifier ces copies sans modifier l'original. La fonction random du module du même nom a rendu notre IA moins prévisible. Enfin, nous avons choisi pygame pour la partie graphique et la récupération des entrées clavier.

II-Présentation générale de l'algorithme

A-Représenter le plateau



Le plateau de Quoridor est représenté par un tableau, c'est-à-dire des listes dans une liste. Ici, les 17 éléments de la liste sont eux-mêmes des listes contenant 17 éléments. Le tableau s'appelle plateau, et pour obtenir le contenu d'une case il suffit d'utiliser la syntaxe plateau[x][y] où x représente la ligne et y la colonne (voir image ci-contre, qui représente le plateau visible) en couverture.

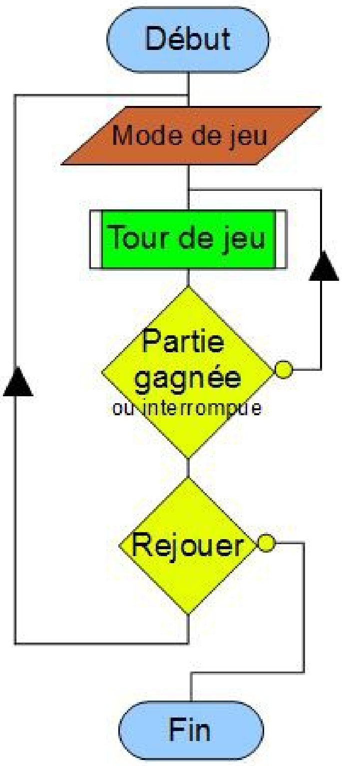
Une barrière se note 'B', le pion du joueur 1 est 'J1' sauf si ce joueur est un IA, dans ce cas il est représenté par 'IA'. Le pion du joueur 2 est 'J2'.

Nous avons opté pour un tableau de 17 par 17 pour représenter le plateau de 9 cases par 9 afin de pouvoir laisser les espaces nécessaires pour écrire les barrières entre les cases où se déplacent les pions. Ainsi, les pions se déplacent de deux cases par deux cases dans ce tableau et se trouvent forcément sur une ligne et une colonne au numéro pair.

A l'inverse, une barrière verticale se situe dans une colonne impaire, une horizontale dans une ligne impaire. Pour qu'une barrière obstrue exatement 2 cases et ne dépasse pas ces 2 cases, il faut la noter sous la forme de 3 'B' alignés et adjacents (hors diagonales). Par conséquent son centre occupe une ligne et colonne impaires.

Dans le cas représenté, on a plateau=[['','','','','','','','','','','','','','','','',''],['','','','','','','','B','B','B','','B','B','B','','',''],['J2','','','','','','B','','','','','','','','','B',''],['','','','','','','B','','','','','','','','','B',''],['','','','','','','B','','','','B','','','','','B',''],['','','B','B','B','','B','B','B','','','B','','','','',''],['','','','','','','','','','','B','','','','','B',''],['B','B','B','','B','B','B','','B','B','B','','','','','B',''],['','','','','','','','','','','','','','','','B',''],['','','','','','','','','','','','B','B','B','','',''],['','','','','','','','','','','B','','','','','',''],['','','','','','','','B','B','B','B','','','','','',''],['','','','','','','','','','','B','','','','','',''],['B','B','B','','B','B','B','','B','B','B','','','','','',''],['','','','','','','J1','','','','','','','','','',''],['','','','','','','B','B','B','','B','B','B','','B','B','B'],['','','','','','','','','','','','','','','','','']].

B-La structure du programme



Au début du programme, on ouvre la fenêtre et trace le plateau de jeu et les pions. On demande ensuite à l'utilisateur s'il désire jouer contre un autre humain sur la même machine ou s'il préfère affronter l'IA. On peut alors créer le tableau contenant le plateau (on sait s'il faut marquer J1 ou IA). On va maintenant faire jouer à tour de rôle les joueurs en mettant à jour les informations utiles comme les touches à utiliser, le nombre de barrières restant à chaque joueur, quel joueur doit faire son choix et l'éventuelle raison du refus d'un coup demandé par un joueur, et ce jusqu'à ce que l'un l'emporte ou clique sur la croix permettant de fermer la fenêtre. Une fois cela fait on proposera au(x) joueur(s) de recommencer une

partie, avec la possiblité de changer de mode de jeu (ajouter ou enlever l'IA). L'utilisateur peut alors choisir de quitter le jeu.

Le code python comprend trois classes principales : Jouer, IA et A_Star (une quatrième classe existe, la classe Node, utilisée exclusivement par la classe A_Star et qui comprend uniquement un constructeur). En dehors de ces classes on trouve quelques fonctions et du code, permettant une partie de la gestion de l'affichage, mais aussi de débuter une partie, d'en recommencer une ou de quitter.

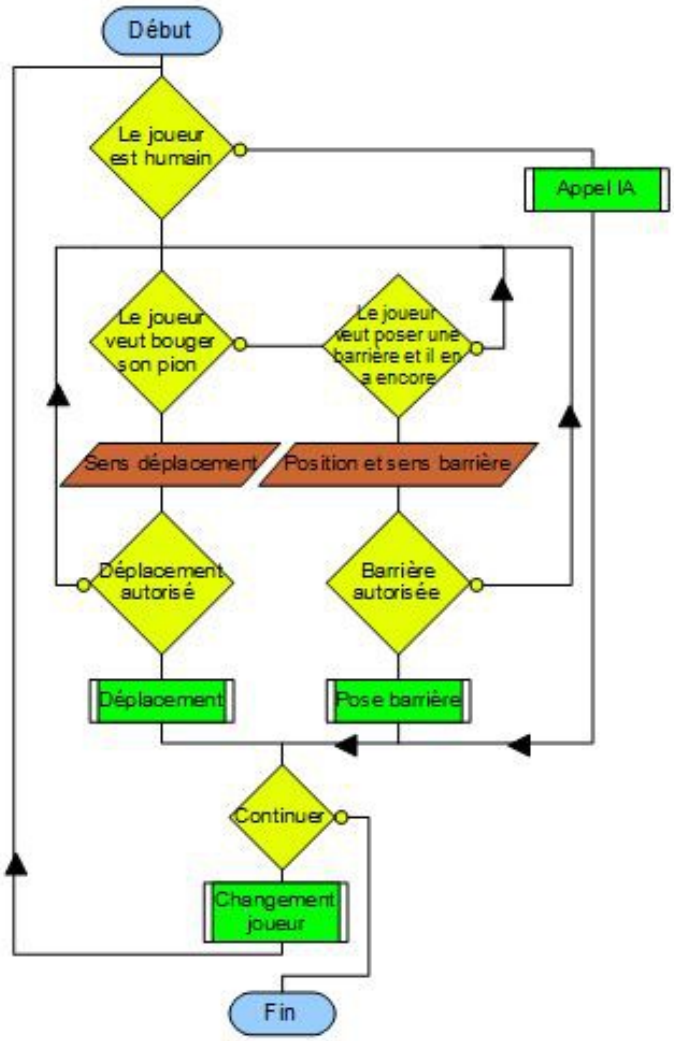
La classe Jouer fait jouer les joueurs à tour de rôle.

La classe IA permet de simuler un joueur en choisissant un coup. Pour en choisir un intéressant, elle va tester une à une toutes les possiblités qui s'offrent à elles, soit 4 déplacements et 128 positions de barrière. Elle éliminera les choix illégaux. Elle débute par les déplacements, en choisissant celui qui rapproche le plus son pion de l'arrivée. Si deux déplacements sont équivalents le hasard les départage. Ensuite, elle cherche la barrière qui augmente le plus le trajet de l'adversaire tout en augmentant au minimum le sien. Enfin, elle compare la meilleure barrière au meilleur déplacement pour trouver le meilleur coup, mais sans prévision à plusieurs coups.

Enfin, la classe A_Star permet de trouver le chemin le plus court entre un pion et sa ligne d'arrivée. Elle est donc utile pour la classe IA pour déterminer le chemin restant à son pion et/ou à celui de son adversaire après chaque hypothétique coup, mais aussi à la classe Jouer pour s'assurer qu'une barrière ne bloque pas totalement le passage d'un pion.

C-Le fonctionnement de la classe Jouer

La méthode principale de la classe Jouer est toursDeJeu(). Son fonctionnement est explicité par l'algorigramme ci-contre.



Pour savoir ce qu'un joueur humain veut faire, on appelle la méthode tour_joueur(), qui va récupérer des entrées claviers et appeler d'autres méthodes. Si le joueur appuie sur la touche 'p' c'est qu'il veut bouger son pion, sinon s'il appuie sur la touche 'b' c'est pour poser une barrière. De la même manière le joueur indique le sens de déplacement souhaité par les flèches directionnelles ou positionne la barrière grâce à celles-ci, utilise 'r' pour tourner de 90° la barrière (passer de l'horizontale à la verticale ou inversement) puis valide par la touche entrée.

Pour vérifier la légalité d'un déplacement, on utilise la méthode deplacementAutorise(). Celle-ci va vérifier que le pion reste sur le plateau, qu'il ne

traverse pas de barrière et qu'il n'arrive pas sur une case déjà occupée. Dans ce dernier cas, le déplacement sera autorisé si un saut est possible. Si le pion peut sauter en conservant la même direction il le fera, sinon on utilisera la méthode sautBarriere().

La méthode permettant d'autoriser la pose d'une barrière est barriereAutorisee(). Son fonctionnement est détaillé page 7.

Pour savoir s'il faut continuer, il faut vérifier 2 choses : personne n'a souhaité arrêté et personne n'a gagné. Dans ce but, la méthode tour_joueur() renvoie 0 si le joueur a voulu interrompre la partie. La méthode gagne compare les positions des pions à celles de leurs lignes d'arrivée pour savoir s'il y a un vainqueur.

Pour le changement de joueur, il suffit de savoir qui vient de jouer, d'où l'intéret de l'attribut joueur et s'il y a un IA, ce qui explique la présence de l'attribut vsIA. Si le joueur 1 ou l'IA vient de jouer c'est forcément au tour du joueur 2. Si le joueur 2 a joué, c'est au tour de l'IA s'il y en a un, sinon c'est au joueur 1. En effet l'IA prend toujours la place du joueur 1.

D-Le principe de fonctionnement de l'algorithme A*

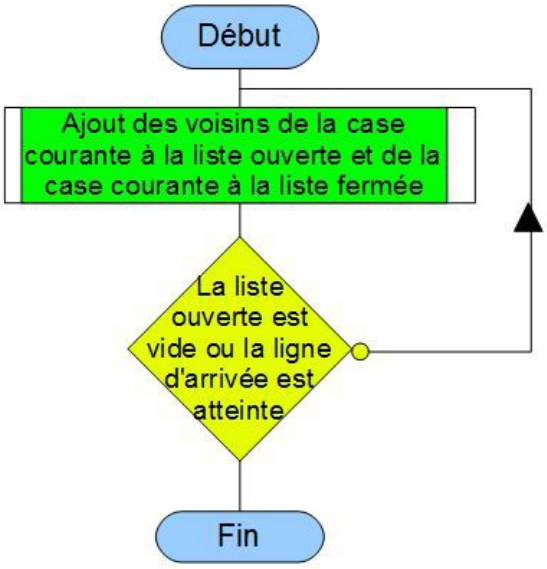
L'algorithme A* (à prononcer A étoile ou A star) est un algorithme de recherche de chemin dans un graphe entre un nœud de départ et un autre d'arrivée. Ici, le graphe est simple puisqu'il s'agit du plateau de Quoridor, les distances entre 2 cases adjacentes (hors diagonales) sont toujours identiques. Dans ces conditions cet algorithme trouve à chaque fois le chemin le plus court entre deux cases quelconques, où que soient placées les barrières.

Son principe de fonctionnement repose sur deux listes.

La première est la liste ouverte. Elle contient les nœuds (ici les cases avec quelques informations supplémentaires comme le nœud parent) qui sont accessibles mais en attente. On n'a pas encore regardé où on pouvait aller à partir de cette case. Les nœuds sont suivis de leurs poids respectifs, c'est-à-dire le nombre de déplacements nécessaires pour arriver à la case plus le nombre de déplacements qu'il faudra dans le meilleur des cas pour atteindre la ligne d'arrivée. En effet dans notre cas, il n'y a pas de nœuds d'arrivée, mais une ligne de nœuds d'arrivée. Cela n'est pas gênant puisque l'on sait que le plus proche, toujours dans le cas idéal, est celui se trouvant sur la même colonne. La distance restante à parcourir est alors l'écart vertical (en valeur absolue) entre la ligne d'arrivée et la case. Sur le tableau, il ne faut pas oublier que l'on se déplace de deux cases à la fois, et donc diviser cet écart par deux.

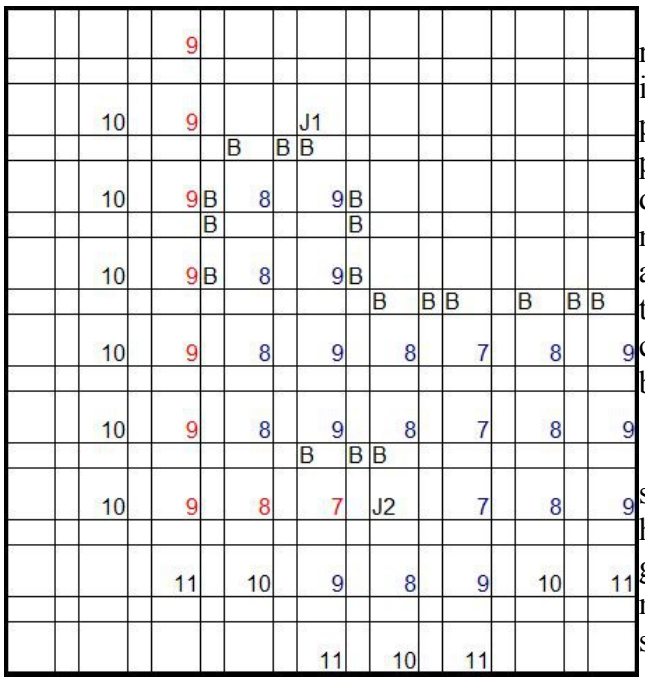
Lorsque l'on a testé tout les voisins d'un nœud de la liste ouverte, on le retire de cette liste et il intègre une seconde liste. Ainsi, si la liste ouverte est vide, cela signifie que toutes les cases atteignables du plateau ont été atteintes. Par conséquent on peut arrêter la recherche et dire qu'il n'y a pas de chemin si aucun nœud de la ligne d'arrivée n'a été atteint entre temps.

La deuxième liste est la liste fermée. Elle contient les nœuds dont on a déjà testé tout les voisins. Ainsi, on pourra l'utiliser pour reconstruire le chemin une fois l'arrivée atteinte.



Le principe de la recherche est visible sur l'algorigramme simplifié ci-contre. Il faut néanmoins clarifier ce qu'est la case courante. A la première itération, il s'agit du nœud de départ. Ensuite, il s'agira du nœud de la liste ouverte qui possède le poids le plus faible.

De plus, lorsque l'on teste les voisins d'un nœud de la liste ouverte, on peut ignorer ceux appartenant à la liste fermée car le nouveau calcul du poids donnerait alors un résultat égal ou supérieur. Par contre, si un voisin appartient déjà à la liste ouverte, il faut tout de même calculer son poids car il peut alors être inférieur. Dans ce cas, on modifie le poids précédement enregistré.



Ci-contre le résultat de la simulation d'une recherche. Les nombres sont les poids des cases, ie le nombre minimal de coups que l'on mettra pour atteindre l'arrivée en supposant qu'il n'y aura pas d'obstacle, et en comptant les déplacements déjà effectués pour arriver à cette case. Les nombres en rouges sont ceux des cases appartenant au chemin le plus cout. En bleu, on trouve celle qui sont dans la liste fermée, en noir celle qui sont restées dans la liste ouverte jusqu'au bout.

Pour l'implémentation python, je me suis basé sur le code disponible sur le forum ci-après: http://www.developpez.net/forums/d794333/general-developpement/algorithme-mathematiques/algorithmes/ameliorer-algorithme-star/

III-Explication détaillée de certaines fonctionnalités

A-La récupération des entrées clavier

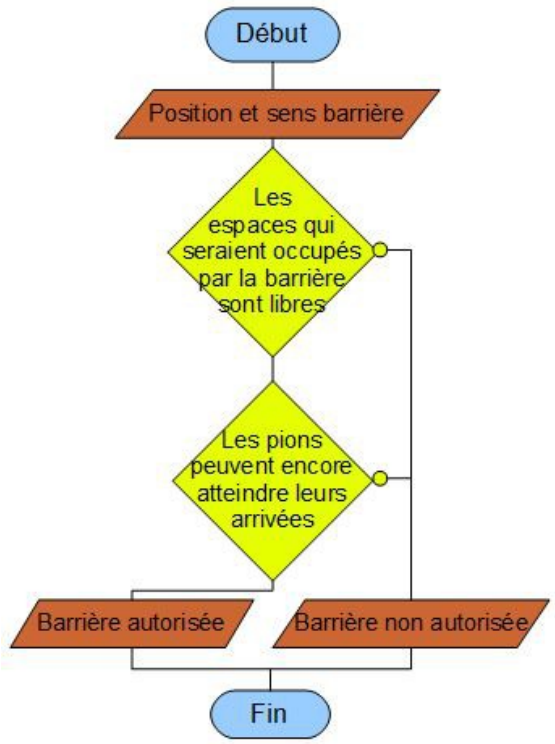
La récupération des entrées clavier s'effecture via pygame. Il s'agit en fait de sauvegarder des événements puis de les traiter un par un. On récupère les événements par pygame.event.get(). Une fois recépurés ils sont automatiquement retirés de la liste des événements enregistrés.

Pour exploiter les événements, on commence par regarder quel type d'événement c'est produit. Vouloir quitter pygame, par exemple par un clic sur la croix pour fermer la fenêtre, est un événement de type QUIT. Les entrées clavier sont disponibles sont enregistrées deux fois : une fois quand la touche est pressée, sous le type KEYDOWN, et une seconde fois lorsqu'elle est relâchée, sous le type KEYUP. On ne traitera que les événements de type QUIT et KEYDOWN.

Parmi les événements de type KEYDOWN, on sélectionnera ceux correspondants aux touches attendues grâce à leur attribut key. Cet attribut commence par la K_ et fini par le nom de la touche. Ainsi l'événement associé à la pression de la touche 'b' a pour attribut key K_b. Celui associé à la touche entrée est K_RETURN.

Il faut penser à intégrer la récupération des entrées clavier dans une boucle afin de laisser à l'utilisateur le temps d'appuyer sur la touche de son clavier. Il est utile de limiter la vitesse de cette boucle par pygame.time.Clock().tick(n), où n représente le nombre maximal de tours de boucle par seconde, afin de ne pas surcharger le processeur.

B-Autoriser la pose d'une barrière



La méthode permettant d'autoriser la pose d'une barrière est encapsulée dans la classe Jouer et se nomme barriereAutorisee() (voir son algorigramme ci-contre). Elle prend en argument la position sur le plateau du centre de la barrière et son sens.

Elle fait appel à la méthode occupe() pour vérifier que la barrière ne se supperpose pas (complètement ou partiellement) à une autre. Pour cela, elle détermine dans quelles positions du tableau plateau les 'B' représentant la barrière seraient écris. Si un 'B' se trouve déjà dans l'une de ces positions, alors cela signifie que la barrière empiéterait sur une autre. Elle est donc non autorisée.

Remarque : un pion ne peut pas se trouver sur une de ces positions puisque l'on pose les barrières uniquement dans les interstices entre les cases où se déplacent les pions.

Elle fait également appel à la méthode bloque() pour vérifier que les deux pions peuvent encore atteindre leurs arrivées respectives malgré cette barrière. Pour cela, on commence par créer une copie du plateau, que l'on pourra modifier sans toucher à l'original, sur laquelle on écrit la barrière comme si elle était autorisée et posée. Ensuite on utilise la méthode pathfinding() (recherche de chemin) de la classe A_Star deux fois, afin de chercher s'il existe encore un chemin entre chacun des pions et leurs lignes d'arrivée. Si ce n'est pas le cas pour au moins un des pions, la barrière n'est pas autorisée.

Remarque : Le fait que ce chemin soit le plus court n'a pas ici d'importance.

Conclusion

Ce projet réalisé en binôme m'aura permis d'apprendre à mieux travailler au sein d'un groupe (très) restreint.

La recherche de solutions tout au long de ce travail a apporté des compléments bienvenus au cours de programmation python dispensés en ISN, notamment en ce qui concerne l'algorithme de recherche du chemin le plus court.

Annexe : le code python

```
# -*- coding: utf-8 -*-

# Projet ISN : Quoridor
# Par Anthony BACCUET et Bastien RAVOT

import pygame # importation du module permettant la communication avec le(s) joueur(s) par l'affichage et la récupération des entrées clavier
from pygame.locals import *

import copy
from copy import deepcopy # récupération de la fonction permettant de copier une liste de façon à ce qu'une modification sur la copie n'entraine pas une modification de l'original

from random import random # récupération de la fonction permettant de générer au hasard un réel compris entre 0 et 1


def initialisationPlateau(vsIA):
    """Fonction permettant de créer une matrice (tableau) de 17 par 17 représentant le plateau de Quoridor.
    Elle prend en argument un booléen valant 1 s'il s'agit d'une partie contre l'IA, 0 sinon.
    Elle retourne la matrice."""

    plateau=['']*17 # création d'une liste de 17 éléments
    for k in range(17):
        plateau[k]=['']*17 # chacun des 17 éléments devient lui-même une liste de 17 éléments
    if vsIA:plateau[0][8]='IA' # écriture des pions à leurs emplacements de départ
    else:plateau[0][8]='P1'
    plateau[16][8]='P2'

    return plateau

def changementCommandes(texte):
    """Fonction permettant d'actualiser les instructions au joueur. Attention cela ne modifie par l'affichage, il faut attendre le prochain pygame.display.update().
    Elle prend en argument les commandes à afficher (un texte)."""

    fenetre.blit(backgroundZone,zone) # on efface les précédentes lignes
    x,y = zone.topleft # on va écrire depuis le coin en haut à gauche de l'emplacement dédié
    for ligne in texte.splitlines(): # pour chaque ligne à écrire
        x,y = fenetre.blit(font.render(ligne,0,(0,0,0)),(x,y)).bottomleft # on écrit la ligne et on récupère le coin en bas à gauche de l'emplacement (rectangle pygame) qu'elle occupe pour pouvoir reprendre l'écriture à ce point et donc écrire les lignes les unes au-dessous des autres (pas de chevauchement)

def changementNbBarrieres(nb1,nb2):
    """Fonction permettant d'actualiser le nombre de barrière(s) restante(s) à chaque joueur. Attention cela ne modifie par l'affichage, il faut attendre le prochain pygame.display.update().
    Elle prend en argument le nombre de barrière(s) restante(s) à chaque joueur."""

    fenetre.blit(background,zoneBarrieres) # même fonctionnement que la précédente fonction
    barrieres="""Nb barrières restantes J1 : {}
Nb barrières restantes J2 : {}""".format(nb1,nb2)
```

```
    x,y = zoneBarrieres.topleft
    for ligne in barrieres.splitlines():
        x,y = fenetre.blit(font.render(ligne,0,(0,0,0)),(x,y)).bottomleft

def fin(gagnant,vsIA):
    """Fonction permettant de dire au(x) joueur(s) qui a gagné et de demander s'il faut recommencer une
partie.
    Elle prend en argument le gagnant (0 si partie interrompue) et le booléen indiquant si un IA joue ou non.
    Elle renvoie 1 si le(s) joueur(s) veut/veulent refaire une partie, 0 sinon."""

    fenetre.blit(background,zone) # on efface les instructions au(x) joueur(s) et le nombre de barrières
restantes à chaque joueur
    if gagnant==0: texte="""Partie interrompue !
F1 pour une nouvelle partie.
F2 pour quitter.""" # s'il n'y a pas de gagnant c'est que la partie a été interrompue (pas de nul possible au
Quoridor)

    elif vsIA and gagnant==1:texte="""L' IA l'emporte !!!
F1 pour une nouvelle partie.
F2 pour quitter.""" # l'IA joue en tant que joueur 1

    else: texte="""Le joueur {} l'emporte !!!
F1 pour une nouvelle partie.
F2 pour quitter.""".format(gagnant)

    changementCommandes(texte) # on annonce le gagnant et demande au(x) joueur(s) leur choix

    fenetre.blit(backgroundJoueur,zoneJoueur) # on remplace la partie de la fenêtre indiquant qui doit jouer
par le fond
    zones=[zone,zoneBarrieres,zoneJoueur]
    pygame.display.update(zones) # on ne met pas à jour tout l'affichage car cela est inutile, le plateau n'a pas
changé

    choix=0
    while not choix: # on récupère le choix
        pygame.time.Clock().tick(30)
        for event in pygame.event.get():
            if event.type == QUIT:
                return 0 # fermeture de fenêtre donc l'utilisateur veut clairement quiter
            elif event.type == KEYDOWN:
                if event.key == K_F1:
                    return 1
                elif event.key == K_F2:
                    return 0

#_____________________________________________________________________

class Jouer:
    """Classe principale, gérant une partie depuis la position de départ jusqu'à la victoire d'un joueur ou
l'interruption de la partie."""

    def __init__(self,joueur,plateau,x_P1,y_P1,x_P2,y_P2,nbBarrieres1,nbBarrieres2,fenetre,vsIA):
        """Fonction qui est appelée lors de l'initialisation de la classe pour initialiser les variables propres à
cette classe."""
```

```
        self.joueur = joueur
        self.plateau = plateau
        self.x_P1 = x_P1
        self.x_P2 = x_P2
        self.y_P1 = y_P1
        self.y_P2 = y_P2
        self.x_P=x_P1
        self.y_P=y_P1
        self.nbBarrieres1 = nbBarrieres1
        self.nbBarrieres2 = nbBarrieres2
        self.nbBarrieres = nbBarrieres1
        self.fenetre=fenetre
        self.joue=0
        self.vsIA=vsIA

    def toursDeJeu(self):
        """Fonction permettant de faire jouer les joueurs et/ ou l'IA chacun leur tour jusqu'à ce qu'il y ait un gagnant ou que l'un des joueurs souhaite arrêter.
        Elle renvoie 0 si la partie a été interrompue, 1 si le joueur 1 ou l'IA a gagné, 2 si c'est le joueur 2."""

        plateauBase=pygame.image.load('fenetre.bmp')
        fenetre.blit(plateauBase,ecran) # on recharge pour effacer la barrière mobile (voir barriereMobile(self,x_B,y_B,sens)) si le joueur est revenu sur son choix de poser une barrière
        pygame.display.update()
        continuer = 1
        while continuer:
            if self.joueur=='P1' or self.joueur=='IA':
                self.nbBarrieres=self.nbBarrieres1 # on utilise ces variables pour ne pas avoir à regarder qui joue à chaque fois
                self.x_P=self.x_P1
                self.y_P=self.y_P1
            else:
                self.nbBarrieres=self.nbBarrieres2
                self.x_P=self.x_P2
                self.y_P=self.y_P2

            if self.joueur=='IA':self.tour_IA() # on utilise des fonctions différentes suivant qu'il s'agisse d'un IA ou d'un joueur physique
            else: continuer=self.tour_joueur() # continuer vaudra 0 si le joueur veut interrompre la partie, on sortira ainsi de la boucle de jeu

            if self.joue: # si le joueur ou IA a fait un choix valide
                pygame.display.update()
                gagnant=self.gagne() # on regarde si ce joueur a gagné
                if gagnant!=0:
                    continuer=0
                    return gagnant # si oui on sort de cette fonction qui fait jouer les joueurs
                self.changementJoueur() # sinon on passe au joueur suivant
                self.joue=0 # ce nouveau joueur n'a pas encore joué

        return 0 # il n'y a pas de gagnant si la boucle est interrompue

    def changementJoueur(self):
        """Fonction permettant d'avertir visuellement que c'est le tour du joueur suivant et de mettre à jour la variable joueur."""
```

```
        if self.joueur=='P1' or self.joueur=='IA': # si c'était le joueur 1 ou l'IA on passe au joueur 2
            self.joueur='P2'
            texteJoueur="Au tour du joueur 2"
        elif self.vsIA==False:
            self.joueur='P1'
            texteJoueur="Au tour du joueur 1"
        else:
            self.joueur='IA'
            texteJoueur="Au tour de l'IA"
        fenetre.blit(backgroundJoueur, zoneJoueur) # suppression de la précedente ligne
        fenetre.blit(font.render(texteJoueur,0,(0,0,0)),(160,10)) # écriture de la nouvelle
        pygame.display.update() # affichage du résultat
        pygame.image.save(fenetre,'fenetre.bmp') # on sauve la nouvelle fenêtre

    def tour_IA(self):
        """Fonction permettant de faire jouer l'IA lors avec de son tours avec la classe IA"""

        ia=IA(self.joueur, self.plateau, self.x_P1, self.y_P1, self.x_P2, self.y_P2, self.nbBarrieres1,
self.nbBarrieres2, self.fenetre, self.vsIA)      #appelle la classe IA
        self.x_P,self.y_P,self.joue,self.plateau,self.nbBarrieres=ia.joue()        #pour faire jouer l'IA
        self.x_P1=self.x_P
        self.y_P1=self.y_P
        self.nbBarrieres1=self.nbBarrieres

        pygame.draw.circle(fenetre,couleurPion1,[70+25*self.y_P,70+25*self.x_P],15)   #affiche le pion de l'IA
à la nouvelle position
        changementNbBarrieres(self.nbBarrieres1,self.nbBarrieres2) #actualise l'affichage du nbre de barrieres

    def tour_joueur(self):
        """Fonction permettant aux joueurs de jouer lorsque c'est leur tour et renvoie 0 si le joueur a joue et 1 si
le joueur a quitte la partie"""

        continuer=1
        auJoueur=1
        while auJoueur:                 #on reste dans la boucle tant que le joueur n'a pas joué
            pygame.time.Clock().tick(30)
            for event in pygame.event.get():
                if event.type == QUIT:      #si le joueur veut quitter, on sort de la boucle
                    continuer = 0
                    auJoueur=0
                elif event.type == KEYDOWN:           #recupere le choix du joueur
                    if event.key == K_p:              #si le joueur appuie sur p
                        self.deplacerPion()           #il deplace le pion
                        if self.joueur=='P1':     #enregistre la position du pion et affiche le pion en fonction du joueur
                            self.x_P1=self.x_P
                            self.y_P1=self.y_P
                            pygame.draw.circle(fenetre,couleurPion1,[70+25*self.y_P,70+25*self.x_P],15)
                        else:
                            self.x_P2=self.x_P
                            self.y_P2=self.y_P
                            pygame.draw.circle(fenetre,couleurPion2,[70+25*self.y_P,70+25*self.x_P],15)
                        auJoueur=0

                    elif event.key == K_b and self.nbBarrieres>0:   #si le joueur appuie sur b et qu'il lui reste des
barrieres
```

```
                self.poserBarriere()                    #on lance la fonction permettant de poser une barriere

                if self.joueur=='P1' or self.joueur=='IA': self.nbBarrieres1=self.nbBarrieres       #on enregistre le changement du nombre de barriere
                else: self.nbBarrieres2=self.nbBarrieres
                changementNbBarrieres(self.nbBarrieres1,self.nbBarrieres2)
                auJoueur=0
        return continuer

    def deplacerPion(self):
        """fonction permettant au joueur de choisir la direction pour deplacer le pion"""

        while not self.joue:
            pygame.time.Clock().tick(30)
            for event in pygame.event.get():
                if event.type == QUIT:              #si le joueur veut quitter, on sort de la boucle
                    fin(0)
                    self.joue=1
                elif event.type == KEYDOWN:         #on recupere la direction que veut choisir le joueur
                    if event.key == K_ESCAPE:
                        self.toursDeJeu()
                    if event.key == K_RIGHT:
                        self.deplacementPion(0,2)   #on lance la fonction permettant de deplacer le pion dans la direction choisie
                    elif event.key == K_LEFT:
                        self.deplacementPion(0,-2)
                    elif event.key == K_UP:
                        self.deplacementPion(-2,0)
                    elif event.key == K_DOWN:
                        self.deplacementPion(2,0)

    def deplacementPion(self,dep_x,dep_y):
        """fonction permettant de deplacer le pion, de tester si le deplacement est autorisé et de verifier si il y a un saut
        prend en arguments le deplacement sur x et sur y du pions
        dans le cas ou l'IA joue, elle renvoie autorise, le deplacement sur x et sur y et le deplacement minimum dans le cas d'un saut ou -1 dans le cas contraire"""

        nb_deplacements_min_saut=-1
        autorise,saut=self.deplacementAutorise(dep_x,dep_y)         #verifie si le deplacement est autorisé et si c'est un configuration de saut
        if saut:
            if self.deplacementAutorise(dep_x*2,dep_y*2,dep_x//2,dep_y//2)==(1,0):          #verifie si c'est un saut normal ou un saut avec une barriere derriere
                dep_x=2*dep_x
                dep_y=2*dep_y
            elif self.joueur=="IA" :            #si c'est le tour de l'IA
                ia=IA(self.joueur, self.plateau, self.x_P1, self.y_P1, self.x_P2, self.y_P2, self.nbBarrieres1, self.nbBarrieres2, self.fenetre, self.vsIA)  #lance le programme de l'IA dans le cas d'un saut avec une barriere
                autorise,dep_x,dep_y,nb_deplacements_min_saut=ia.sautbar(dep_x,dep_y)
            else:
                autorise,dep_x,dep_y=self.sautBarriere(dep_x,dep_y)         #lance le programme de saut avec barriere pour le joueur
        if autorise and self.joueur!="IA":      #si ce n'est pas l'IA qui joue, affiche la nouvelle position du pion et modifie le plateau et les positions du pion
```

```
            self.plateau[self.x_P][self.y_P]=''
            pygame.draw.circle(fenetre,couleurCase,[70+25*self.y_P,70+25*self.x_P],20)
            self.x_P+=dep_x
            self.y_P+=dep_y
            self.plateau[self.x_P][self.y_P]=self.joueur
            self.joue=1
        elif self.joueur=="IA":
            return autorise,dep_x,dep_y,nb_deplacements_min_saut

    def sautBarriere(self,dep_x,dep_y):
        """fonction permettant au joueur de choisir la direction qu'il veut prendre dans le cas ou le pion doit
sauter par dessus l'autre pion avec une barriere et
        permet de verifier si le deplacement est autorisé
        prend en argument le deplacement sur x et sur y
        et renvoie autorise et le deplacement sur x et y"""

        fait=0
        texte = """Vous allez sauter par dessus
votre adversaire mais
une barrière est derrière lui.
Choisissez une direction
vers laquelle sauter."""
        changementCommandes(texte)              #affiche le texte precedent
        pygame.display.update(zone)
        changementCommandes(texteCommandes)

        choisi=0
        while not choisi:                   #recupere le choix du joueur
            pygame.time.Clock().tick(30)
            for event in pygame.event.get():
                if event.type == QUIT: fin(0)
                elif event.type == KEYDOWN:
                    if event.key == K_ESCAPE:
                        self.toursDeJeu()
                        choisi=1

                    if event.key == K_RIGHT:
                        if dep_y!=0: autorise=self.refus_saut() #si le deplacement n'est pas possible on l'indique au
joueur
                        else:
                            autorise,saut=self.deplacementAutorise(dep_x,2,dep_x//2) #sinon, verifie si le deplacement
est autorise
                            if autorise:dep_y+=2
                        choisi=1

                    elif event.key == K_LEFT:                                   #meme chose qu'au dessus
                        if dep_y!=0: autorise=self.refus_saut()
                        else:
                            autorise,saut=self.deplacementAutorise(dep_x,-2,dep_x//2)
                            if autorise:dep_y-=2
                        choisi=1

                    elif event.key == K_UP:                                     #meme chose qu'au dessus
                        if dep_x!=0: autorise=self.refus_saut()
```

```
                else:
                    autorise,saut=self.deplacementAutorise(-2,dep_y,0,dep_y//2)
                    if autorise:dep_x-=2
                choisi=1
            elif event.key == K_DOWN:                                   #meme chose qu'au dessus
                if dep_x!=0: autorise=self.refus_saut()
                else:
                    autorise,saut=self.deplacementAutorise(2,dep_y,0,dep_y//2)
                    if autorise:dep_x+=2
                choisi=1

        return autorise,dep_x,dep_y

    def refus_saut(self):
        """permet d"afficher que le deplacement est interdit et renvoie 0"""

        texte="""Vous ne pouvez pas sauter
dans cette direction
(sur-place interdit)
Choisissez une autre direction"""
        changementCommandes(texte)                      #affiche le texte precedant
        pygame.display.update(zone)
        changementCommandes(texteCommandes)
        return 0

    def deplacementAutorise(self,dep_x,dep_y,dep_b_x=0,dep_b_y=0):
        """verifie si le deplacement ne fait pas sortir le pion du plateau
        prend en entré le deplacement sur y et sur x et dep_b_x et dep_b_y dans le cas ou on a un saut et qui
valent 0 dans le cas contraire
        et renvoie 1 ou 0 si le de placement est autorisé ou non et 1 ou 0 si il y a un saut ou non"""

        x_fin=self.x_P+dep_x            #calcule la position du pion apres le mouvement
        y_fin=self.y_P+dep_y
        if x_fin>16 or x_fin<0 or y_fin>16 or y_fin<0: #verifie si cette position est encore sur le plateau de jeu
            if self.joueur!='IA':                       #si ce n'est pas un IA
                texte = """Vous souhaitez sortir du plateau !
Choisissez une autre direction"""
                changementCommandes(texte)              #previent le joueur que son pion sort du plateau
                pygame.display.update(zone)

                changementCommandes(texteCommandes)
            return 0,0

        x=self.x_P+(dep_x//2)+dep_b_x  #calcule la position que traverse le pion (position ou peut se situer une
barriere)
        y=self.y_P+(dep_y//2)+dep_b_y
        if self.plateau[x][y]=='B':                     #verifie si on traverse une barriere
            if self.joueur!='IA':                       #et previent le joueur si c'est le cas
                texte = """Vous souhaitez traverser une barrière !
Choisissez une autre direction"""
                changementCommandes(texte)
                pygame.display.update(zone)
                changementCommandes(texteCommandes)
            return 0,0
```

```
        if self.plateau[x_fin][y_fin]=='P1' or self.plateau[x_fin][y_fin]=='P2' or self.plateau[x_fin]
[y_fin]=='IA':       #verifie si il y a un pion sur la position d'arrivéeet donc un saut
            return 1,1

        return 1,0

    def poserBarriere(self):
        """fonction permettant aux joueurs de poser une barriere"""

        pygame.image.save(fenetre,'fenetre.bmp')
        x_B=9               #positionne au depart la barriere au centre du plateau de jeu et horizontalement
        y_B=9
        sens='h'
        self.barriereMobile(x_B,y_B,sens)          #affiche la barriere
        fait=0
        while not fait:
            pygame.time.Clock().tick(30)
            for event in pygame.event.get():
                if event.type == QUIT:
                    fin(0)
                    self.joue=1
                elif event.type == KEYDOWN:
                    if event.key == K_ESCAPE:   #si le joueur veut choisir autre chose on relance la fonction de
choix
                        self.toursDeJeu()
                        fait=1

                    elif event.key == K_RIGHT and y_B<15:          #deplace la barriere dans la direction voulue si
ça ne le fait sortir du plateau de jeu
                        y_B+=2
                        self.barriereMobile(x_B,y_B,sens)          #reaffiche la barriere à la nouvelle position
                    elif event.key == K_LEFT and y_B>1:
                        y_B-=2
                        self.barriereMobile(x_B,y_B,sens)
                    elif event.key == K_UP and x_B>1:
                        x_B-=2
                        self.barriereMobile(x_B,y_B,sens)
                    elif event.key == K_DOWN and x_B<15:
                        x_B+=2
                        self.barriereMobile(x_B,y_B,sens)
                    elif event.key == K_r:                      #permet de changer le sens de la barriere
                        if sens=='h':sens='v'
                        else: sens='h'
                        self.barriereMobile(x_B,y_B,sens)          #reaffiche la postion de la barriere

                    if event.key == K_RETURN:   #valide la position de la barriere lorsque le joueur appuie sur
entrée
                        self.poseBarriere(x_B,y_B,sens)
                        fait=1


    def barriereMobile(self,x_B,y_B,sens):
        """fonction permettant d'afficher la barriere pendant que le joueur la deplace pour la placer
        prend en argument la position de la barriere sur x et y et on sens"""
```

```
        x_Bc=70+25*x_B            #calcule la position ou afficher la barriere
        y_Bc=70+25*y_B
        if sens=='h':             #calcule les coordonées exactes en fonction du sens
            gauche=y_Bc-45
            haut=x_Bc-5
            largeur=90
            hauteur=10
        else:
            gauche=y_Bc-5
            haut=x_Bc-45
            largeur=10
            hauteur=90
        plateauBase=pygame.image.load('fenetre.bmp')              #affiche la barriere sur le plateau de jeu
        fenetre.blit(plateauBase,ecran)
        pygame.draw.rect(fenetre,couleurBarriereMobile,[gauche,haut,largeur,hauteur])
        pygame.display.update(plateauAffiche)

    def poseBarriere(self,x_B,y_B,sens):
        """fonction permetant de modifier le plateau, la matrice et les vriables apres la pose d'une barriere si elle est autorisé
        prend en argument la position sur x et y et le sens de la barriere"""

        autorise=self.barriereAutorisee(x_B,y_B,sens)          #verifie que la pose de la barriere est autorisée
        if autorise:
            self.plateau=self.ecritBarriere(x_B,y_B,sens)   #modifie la matrice avec la nouvelle position de la barriere
            self.afficheBarriere(x_B,y_B,sens)          #modifie l'affichage avec la nouvelle position de la barriere
            self.joue=1
            self.nbBarrieres-=1

    def afficheBarriere(self,x_B,y_B,sens):
        """fonction permettant d'afficher la barriere
        prend en argument la position sur x et y et le sens de la barriere"""

        plateauBase=pygame.image.load('fenetre.bmp')
        fenetre.blit(plateauBase,ecran)
        x_B=70+25*x_B             #calcule la position ou afficher la barriere
        y_B=70+25*y_B
        if sens=='h':             #calcule les coordonées exactes en fonction du sens
            gauche=y_B-45
            haut=x_B-5
            largeur=90
            hauteur=10
        else:
            gauche=y_B-5
            haut=x_B-45
            largeur=10
            hauteur=90
        pygame.draw.rect(fenetre,couleurBarriere,[gauche,haut,largeur,hauteur])        #affiche la barriere a la position definie par le joueur

    def barriereAutorisee(self,x_B,y_B,sens):
        """fonction permettant de savoir si le joueur est autorise a poser la barriere
        prend en argument la position sur x et sur y et le sens
        renvoie 1 sila pose est autorise et 0 dans le cas contraire"""
```

```
        if self.occupe(x_B,y_B,sens) or self.bloque(x_B,y_B,sens):     #verifie que la barriere ne bloque pas un des joueurs ou n'est pas placé sur une autre barriere
            if self.joueur!="IA":
                plateauBase=pygame.image.load('fenetre.bmp')      #modife l'affichage
                fenetre.blit(plateauBase,ecran)
                pygame.display.update(plateauAffiche)
            return 0
        return 1

    def occupe(self,x_B,y_B,sens):
        """fonction permettant de verifier sila case ou le joueur veut poser la barriere n'est pas deja occupé par une autre barriere
        prend en argument la position de la barriere sur x et y et son sens
        renvoie 1 si la case est occupé et renvoie 0 dans le cas contraire"""

        occupe=0
        if sens=='v':
            for k in range(-1,2):     #verifie qu'il n'y a pas de barriere deja placé sur la position de la nouvelle barriere dans le cas d'une barriere verticale
                if self.plateau[x_B+k][y_B]=='B':occupe=1
        else:
            for k in range(-1,2):   #verifie qu'il n'y a pas de barriere deja placé sur la position de la nouvelle barriere dans le cas d'une barriere horizontale
                if self.plateau[x_B][y_B+k]=='B':occupe=1
        if occupe and self.joueur!="IA":
            texte = """Vous ne pouvez pas
poser de barrière ici.
Une autre y est déjà"""
            changementCommandes(texte)    #avertit le joueur qu'il ne peut pas poser de barriere si une barriere est deja posé a cet endroit
            pygame.display.update(zone)

            changementCommandes(texteCommandes)
        return occupe

    def bloque(self,x_B,y_B,sens):
        """fonction permettant de verifier si la pose de la barriere ne bloque un des deux joueurs
        prend en argument la position sur x et y et le sens de la barriere
        renvoie 1 si la barriere bloque un des 2 joueurs et renvoie 0 dans le cas contraire"""

        plateauTest=self.ecritBarriere(x_B,y_B,sens)      #crée un plateau de jeu avec la barriere de placé
        test1=A_Star([self.x_P1,self.y_P1],16,plateauTest)
        existe1,chemin=test1.pathfinding()        #verifie qu'il existe un chemin permettant de gagner pour chacun des deux joueur
        test2=A_Star([self.x_P2,self.y_P2],0,plateauTest)
        existe2,chemin=test2.pathfinding()
        bloque = not(existe1) or not(existe2)
        if bloque and self.joueur!='IA':
            texte = """Vous ne pouvez pas
poser de barrière ici.
Cela bloque un pion."""
            changementCommandes(texte)   #avertit le joueur qu'il ne peut pas poser de barriere si cela bloque un des deux joueurs
            pygame.display.update(zone)
```

```
            changementCommandes(texteCommandes)
        return bloque

    def ecritBarriere(self,x_B,y_B,sens):
        """fonction permettant de modifier la matrice pour ajouter la barriere à la position choisie
        prend en entrée la position et le sens de la barriere
        renvoie le plateau modifié"""

        nvPlateau=copy.deepcopy(self.plateau)                    #cree une nouvelle martice pour la moifier
        if sens=='v':
            for k in range(-1,2):                            #ajoute la barriere à la nouvelle matrice
                nvPlateau[x_B+k][y_B]='B'
        else:
            for k in range(-1,2):
                nvPlateau[x_B][y_B+k]='B'
        return nvPlateau

    def gagne(self):
        if self.x_P1 == 16 : return 1                        #verifie si un des deux joueurs a gagner
        if self.x_P2 == 0 : return 2
        return 0

#_____________________________________________________________________________


class IA :
    """classe permettant de jouer contre un IA
    on cherche si il est plus interressant de poser une barrière ou de deplacer le pion en comparant les rapport
    entre le chemin le plus court de l'IA et du joueur"""

    def __init__(self,joueur,plateau,x_P1,y_P1,x_P2,y_P2,nbBarrieres1,nbBarrieres2,fenetre,vsIA):
        """fonction appelé lors de l'initialisation de la classe pour initialiser les variables"""

        self.joueur = joueur
        self.plateau = plateau
        self.x_P1 = x_P1
        self.x_P2 = x_P2
        self.y_P1 = y_P1
        self.y_P2 = y_P2
        self.nbBarrieres1 = nbBarrieres1
        self.nbBarrieres2 = nbBarrieres2
        self.fenetre=fenetre
        self.vsIA=vsIA

    def joue(self):
        """ fonction appelé lorsque c'est le tour de l'IA qui trouve le coup le plus interressant et renvoie la
position
        du pion de l'IA, le plateau de jeu et le nombre de barrriere de l'IA"""

        if(self.x_P1==6 and self.y_P1==8 and self.y_P2==8 and self.x_P2==10 and self.nbBarrieres1==10
and self.nbBarrieres2==10): #dans le cas ou aucune barriere n'a été posé et que le joueur et l'IA n'ont fait
qu'avancer tout droit,l'IA place un barriere derierre lui au 4eme tour de jeu
            x_B_max=5       #definie le sens et la position de la barriere
            y_B_max=7
            sens_max='h'
```

```
            essai=Jouer(self.joueur, self.plateau, self.x_P1, self.y_P1, self.x_P2, self.y_P2, self.nbBarrieres1,
self.nbBarrieres2, self.fenetre, self.vsIA) #rappelle la classe jouer
            essai.afficheBarriere(x_B_max,y_B_max,sens_max)  #pour afficher la barriere
            self.plateau=essai.ecritBarriere(x_B_max,y_B_max,sens_max) #et modifier la matrice de jeu
            self.nbBarrieres1-=1 #on change la variable contenant le nombre de barriere de l'IA

        else:
            nb_deplacements_min=74

            test=A_Star([self.x_P2,self.y_P2],0,self.plateau) #on appelle la classe A_star
            existe,chemin=test.pathfinding() #recherche de la longueur du chemin le plus court pour le joueur et
ensuite la comparer à celle des chemins de l'IA après avoir joué
            nb_deplacements_adv=len(chemin)
            nb_deplacements_min,dep_x,dep_y=self.essai_dep(nb_deplacements_min,0,0,-2,0)       #on trouve
la direction pour laquelle le chemin est le plus court et sa longueur
            nb_deplacements_min,dep_x,dep_y=self.essai_dep(nb_deplacements_min,dep_x,dep_y,2,0)
            nb_deplacements_min,dep_x,dep_y=self.essai_dep(nb_deplacements_min,dep_x,dep_y,0,-2)
            nb_deplacements_min,dep_x,dep_y=self.essai_dep(nb_deplacements_min,dep_x,dep_y,0,2)

            if nb_deplacements_min!=0:avance_mouv=nb_deplacements_adv/nb_deplacements_min # si l'IA n'a
pas gagné au prochain coup, on calcul son avantage a deplacer son pion
            else:avance_mouv=100
            avance_test_max=0
            allonge_max=0
            if self.nbBarrieres1>0:        #si il reste des barrieres à l'IA, on calcule l'avantage de poser une barriere

avance_test_max,x_B_max,y_B_max,sens_max,allonge_max=self.av_pose(nb_deplacements_adv)


affichage=Jouer(self.joueur,self.plateau,self.x_P1,self.y_P1,self.x_P2,self.y_P2,self.nbBarrieres1,self.nbBarr
ieres2,self.fenetre,self.vsIA)        #on appelle la classe jouer


            if((avance_mouv<avance_test_max and allonge_max>2 and self.nbBarrieres2-self.nbBarrieres1<=3)
or (nb_deplacements_adv==1 and nb_deplacements_min!=0)) and self.nbBarrieres1!=0 :        #si il reste
des barriere à l'IA, qu'il n'a pas gagné au prchain coup et que le joueur ggne auprochain coup ou que l'interet
a poser et superieur à celui du deplacement, il pose une barriere
                affichage.afficheBarriere(x_B_max,y_B_max,sens_max)        #affiche et enregistre la posiion du
pion sur la matrice avec des fonction de jouer
                self.plateau=affichage.ecritBarriere(x_B_max,y_B_max,sens_max)
                self.nbBarrieres1-=1

            else:              #sinon il deplace son pion sur la matrice et l'affiche
                self.plateau[self.x_P1][self.y_P1]=''
                pygame.draw.circle(fenetre,couleurCase,[70+25*self.y_P1,70+25*self.x_P1],20)
                self.x_P1+=dep_x
                self.y_P1+=dep_y
                self.plateau[self.x_P1][self.y_P1]='IA'

        return self.x_P1,self.y_P1,1,self.plateau,self.nbBarrieres1

    def essai_dep(self,nb_deplacements_min,x_dep,y_dep,dep_x,dep_y):
        """calcul la longueur minimal du chemin en en entrée en fonction du deplacement sur x et sur y et
egalement du chemin le plus court pour les direction
        deja testé et renvoie le nombre de deplacement minimum et le deplacement sur x et y"""
```

```
        essai=Jouer(self.joueur, self.plateau, self.x_P1, self.y_P1, self.x_P2, self.y_P2, self.nbBarrieres1,
self.nbBarrieres2, self.fenetre, self.vsIA)        #on appelle la classe jouer
        autorise, dep_x, dep_y,nb_deplacements_min_saut = essai.deplacementPion(dep_x,dep_y)
        if autorise and nb_deplacements_min_saut==-1:#si le deplacement est autorisé et qu'il n'y a pas de saut,
            test=A_Star([self.x_P1+dep_x,self.y_P1+dep_y],16,self.plateau)#on appelle la classe A_star
            existe,chemin=test.pathfinding()#pour trouver le chemin le plus court
            nb_deplacements=len(chemin)
            if nb_deplacements<=nb_deplacements_min+random()-0.5:       #si le chemin le plus court est plus
court que le precedant (le random permet d'ajouter du hasard dans le cas oules deux font la meme longeur)
                return nb_deplacements,dep_x,dep_y       #on renvoie la longueur du nouveau chemin et le
deplacement du pion
            return nb_deplacements_min,x_dep,y_dep      #sinon on renvoie la longeur precedente avec le
deplacement precedent
        if nb_deplacements_min_saut<=nb_deplacements_min+random()-0.5 and nb_deplacements_min_saut!
=-1:return nb_deplacements_min_saut,dep_x,dep_y      #on fait la meme chose dans le cas ou il y a un saut
        return nb_deplacements_min,x_dep,y_dep

    def sautbar(self,dep_x,dep_y):
        """dans le cas ou le pion doit sauter par dessus un pion avec une barriere derriere, on regarde la
direction la plus interressante en fonction du deplacement sur x et sur y"""

        nb_deplacements_min_saut=73
        ok=0
        essai=Jouer(self.joueur, self.plateau, self.x_P1+dep_x, self.y_P1+dep_y, self.x_P2, self.y_P2,
self.nbBarrieres1, self.nbBarrieres2, self.fenetre,self.vsIA)        #on appelle la classe Jouer
        if essai.deplacementAutorise(0,-2)==(1,0) and dep_x:                #si le deplacement est autorisé
            meilleur,nb_deplacements_min_saut=self.test_sautbar(dep_x,-2,nb_deplacements_min_saut)
#teste le nombre de deplacement minimum
            if meilleur:dep_y=-2  #si le deplacement est plus interessant que le precedent, on sauve le
deplacement du pion

        if essai.deplacementAutorise(0,2)==(1,0) and dep_x:  #les autres directions fonctionnent de la meme
maniere
            meilleur,nb_deplacements_min_saut=self.test_sautbar(dep_x,2,nb_deplacements_min_saut)
            if meilleur:dep_y=2

        if essai.deplacementAutorise(-2,0)==(1,0) and dep_y:
            meilleur,nb_deplacements_min_saut=self.test_sautbar(-2,dep_y,nb_deplacements_min_saut)
            if meilleur:dep_x=-2

        if essai.deplacementAutorise(2,0)==(1,0) and dep_y:
            meilleur,nb_deplacements_min_saut=self.test_sautbar(2,dep_y,nb_deplacements_min_saut)
            if meilleur:dep_x=2

        if dep_x and dep_y:ok=1      #si un des deplacement est autorisé, on renvoie 1 etle deplacement du pion
et le nombre de deplacement minimum
        else:ok=0
        return ok,dep_x,dep_y,nb_deplacements_min_saut

    def test_sautbar(self,dep_x,dep_y,nb_deplacements_min_saut):
        """ on regarde le nombre de deplacement minimum pour une direction lors d'un saut avec une barriere
        derriere et on le compare avec la longueur minimale pour les tests deja effectué en prenant en entrée
        le deplacement sur et y et le nombre mimimum de mouvement des tests deja effectué et renvoie 1 si
        le chemin est plus court que les precedent et 0 dans le cas contraire et egalement le nombre de
deplacement minimum"""
```

```
        meilleur=0
        test=A_Star([self.x_P1+dep_x,self.y_P1+dep_y],16,self.plateau)      #on appelle la classe A_star
        existe,chemin=test.pathfinding()                                    #calcule la longueur du chemin le plus court
        nb_deplacements=len(chemin)
        if nb_deplacements<nb_deplacements_min_saut+random()-0.5:           #si la longueur du chemin est plus courte que celle du precedent ( lerandom a la meme utilité que dans la fonction essai_dep)
            nb_deplacements_min_saut=nb_deplacements                        #on renvoie 1 et la longueur minimum
            meilleur=1
        return meilleur,nb_deplacements_min_saut                            #sinon on renvoie 0

    def av_pose(self,nb_adv_avant):
        """on regarde l'avantage maximal possible en posant une barriere en testant toutes les possibilités en prenant en entrée l'avantage actuel
        et en renvoyant la position de la barriere, son sens, l'avantage à poser la barriere et l'avantage par rapport au coup precedent"""

        test=A_Star([self.x_P1,self.y_P1],16,self.plateau)
        existe,chemin=test.pathfinding()
        nb_avant=len(chemin)

        avance_test_max=-1
        for k in range(8):      #les deux boucle for de 0 à 7 permettent de passer par toutes les positions possibles
            for j in range(8):
                for s in range(2):                  #la troisieme boucle for permet de tester les deux sens
                    x_B=1+2*k
                    y_B=1+2*j
                    if s==0:sens="h"
                    else: sens="v"
                    essai=Jouer(self.joueur, self.plateau, self.x_P1, self.y_P1, self.x_P2, self.y_P2, self.nbBarrieres1, self.nbBarrieres2, self.fenetre, self.vsIA)#on appelle la classe jouer
                    if essai.barriereAutorisee(x_B,y_B,sens):           #si la barriere est autorisé,
                        plateau_test=essai.ecritBarriere(x_B,y_B,sens)  #on crée une matrice de test avec la barriere posé qui permet de tester le nombre de deplacement minimum de l'IA et du joueur
                        test=A_Star([self.x_P1,self.y_P1],16,plateau_test)
                        existe,chemin=test.pathfinding()
                        nb_deplacements=len(chemin)
                        test=A_Star([self.x_P2,self.y_P2],0,plateau_test)
                        existe,chemin=test.pathfinding()
                        nb_deplacements_adv=len(chemin)
                        if nb_deplacements!=0:avance_test=nb_deplacements_adv/nb_deplacements-random()/1000 #on calcule le rapport entre le nombre de deplacement du joueur et celui de l'IA (le random a toujours la meme utilité)
                        else:avance_test=0
                        if avance_test>avance_test_max:   #si le rapport est superieur au precedent rapport maximum,
                            avance_test_max=avance_test           #on enregistre le nouveau et la position de la barriere
                            x_B_max=x_B
                            y_B_max=y_B
                            sens_max=sens
                            allonge_max=nb_deplacements_adv-nb_adv_avant + nb_avant - nb_deplacements #l'allonge permet de comparer l'avantage gagné par rapport au tour precedent
        return avance_test_max,x_B_max,y_B_max,sens_max,allonge_max

#_____________________________________________________________________
```

```
class Node:
    """Un node est l'entité utilisée par l'algorithme A* pour trouver le chemin"""

    def __init__(self,position=[],dis=0,vol=0,parent=[]):
        """Fonction qui est appelée lors de l'initialisation de la classe."""

        self.distance=dis # distance déjà parcourue
        self.vol_oiseau=vol # distance restante à vol d'oiseau
        self.pos_x=position[0]
        self.pos_y=position[1]
        self.parent=parent # node précédent sur le chemin


class A_Star:
    """Algorithme de pathfinding adapté uniquement pour le Quoridor
    On cherche le chemin le plus court dans un tableau 17*17. Le pion se déplace de 2 cases par 2 cases, pas en diagonale.
    Il ne peut pas "survoler" les cases qui contiennent 'B'.
    On s'appuie sur 2 listes de Nodes : la liste ouverte et la liste fermée.
    La liste ouverte contient les Nodes que l'on peut atteindre et dont on a pas exploré tout les voisins et leurs poids respectifs.
    La liste fermée contient les Nodes entièrement testés et leurs poids respectifs."""

    def __init__(self,debut_xy,fin,map):
        """Fonction qui est appelée lors de l'initialisation de la classe."""

        self.liste_ouverte = [] # aucun Node n'a été atteint ni testé
        self.liste_fermee = []
        self.depart= Node(debut_xy,0,0,debut_xy) #node de départ
        self.fin=fin #c'est la ligne d'arrivée
        self.niveau=map # c'est le plateau de Quoridor sous la forme d'une matrice 17*17 (tableau dans tableau) 'B' signifie qu'il y a une barrière

    def pathfinding(self):
        """On active la recherche d'un chemin par cette fonction
        Elle renvoie un booléen indiquant l'extistence d'un chemin et ce chemin s'il existe, [] sinon.
        Attention, [] peut aussi vouloir dire que le point de départ se situe sur la ligne d'arrivée"""

        if self.depart.pos_x==self.fin:return 1,[] # s'il on est déjà arrivé, il y a un chemin, de longueur 0
        self.liste_ouverte.append([self.depart,self.calcul_vol_oiseau(self.depart.pos_x)]) # sinon on va tester le seul Node qu'on est sûr de pouvoir atteindre : le Node de départ. Son poids vaut la distance à vol d'oiseau jusqu'à l'arrivée
        self.initialisation() # on teste ses voisins
        self.enlever_node_liste_ouverte(self.depart) # on le retire de la liste ouverte car tout ses voisins on été testés
        self.liste_fermee.append([self.depart,self.calcul_vol_oiseau(self.depart.pos_x)]) # il a déjà été testé donc on l'ajoute à la liste fermée

        reussite=0 # 0 si pas de chemin, 1 sinon

        #Tant qu'il y a des nodes à tester
        while(len(self.liste_ouverte)>0): # tant qu'il y a des Nodes atteignables dont les voisins n'ont peut-être pas été testés
            node_actuel=self.trouver_poids_min() # on récupère dans la liste ouverte le Node qui est sur le chemin le plus court en supposant qu'il n'y aura pas de nouvel obstacle, ainsi que son poids
```

```
            self.enlever_node_liste_ouverte(node_actuel[0]) # on l'enlève de la liste ouverte
            self.test_node_adjacent(node_actuel) # on teste ses voisins
            self.liste_fermee.append([node_actuel[0],node_actuel[1]]) # pour pouvoir l'ajouter à la liste fermée

            fini, y_fin=self.fini() # on cherche si l'on a rajouté le node final dans la liste fermee
            if(fini):
                reussite=1 # il y a un chemin
                self.fin=[self.fin,y_fin]
                node=self.get_node_liste_fermee(self.fin)
                self.chemin=self.construire_chemin(node)
                break
        if reussite==0:self.chemin=[] # il n'y a pas de chemin si reussite vaut 0
        return reussite,self.chemin

    def initialisation(self):
        """Fonction appelée pour ajouter les premiers Nodes accessibles"""

        if self.depart.pos_x!=0: #test du node en haut
            if(self.niveau[self.depart.pos_x-1][self.depart.pos_y]!='B'):
                self.preparer_ajouter_liste_ouverte(-2,0) # s'il on a le droit d'aller sur ce Node on l'ajoute

        if self.depart.pos_x!=16: #test du node en bas
            if(self.niveau[self.depart.pos_x+1][self.depart.pos_y]!='B'):
                self.preparer_ajouter_liste_ouverte(2,0)

        if self.depart.pos_y!=0: #test du node à gauche
            if(self.niveau[self.depart.pos_x][self.depart.pos_y-1]!='B'):
                self.preparer_ajouter_liste_ouverte(0,-2)

        if self.depart.pos_y!=16: #test du node à droite
            if(self.niveau[self.depart.pos_x][self.depart.pos_y+1]!='B'):
                self.preparer_ajouter_liste_ouverte(0,2)

    def preparer_ajouter_liste_ouverte(self,decalage_x,decalage_y):
        """Fonction appelée par initialisation() pour ajouter les Nodes à la liste ouverte.
        Elle prend en argument les déplacements depuis le Node de départ"""

        position_node=[self.depart.pos_x+decalage_x,self.depart.pos_y+decalage_y] # calcul des nouvelles
coordonnées
        parent=[self.depart.pos_x,self.depart.pos_y] # le parent est le Node de départ
        vol_oiseau=self.calcul_vol_oiseau(self.depart.pos_x+decalage_x)
        distance=1 # on a effectué 1 déplacement
        poids=vol_oiseau+distance
        node=Node(position_node,distance,vol_oiseau,parent)
        self.liste_ouverte.append([node,poids])

    def test_node_adjacent(self,node_actuel):
        """Fonction appelée par pathfinding() pour tester les voisins d'un Node de la liste ouverte.
        Elle prend en argument ce Node."""

        if(node_actuel[0].pos_x!=0): # test node au-dessus du node actuel
            self.test_node(node_actuel,-2,0)

        if(node_actuel[0].pos_x!=16): # test node au-dessous du node actuel
            self.test_node(node_actuel,2,0)
```

```
        if(node_actuel[0].pos_y!=0): #test node à gauche du node actuel
            self.test_node(node_actuel,0,-2)

        if(node_actuel[0].pos_y!=16): #test node à droite du node actuel
            self.test_node(node_actuel,0,2)


    def test_node(self,node_actuel,decalage_x,decalage_y):
        """Fonction appelée par test_node_adjacent() pour ajouter avec les bons arguments un Node (s'il est accessible) à la liste ouverte.
        Elle prend en argument le Node parent et les déplacements depuis ce Node."""

        x=node_actuel[0].pos_x+decalage_x # calcul des nouvelles coordonnées
        y=node_actuel[0].pos_y+decalage_y
        if(self.niveau[x-(decalage_x//2)][y-(decalage_y//2)]!='B'): # si pas de barrière gênante
            test=self.chercher_node_liste_xy(self.liste_fermee,x,y) # on cherche le node dans la liste fermee
            if(test!=1): # s'il n'y est pas
                boole=self.chercher_node_liste_xy(self.liste_ouverte,x,y) # on le cherche dans la liste ouverte
                if(boole==0): #s'il n'y est pas, on va le rajouter
                    distance=1+node_actuel[0].distance # on a effectué un déplacement de plus
                    vol=self.calcul_vol_oiseau(x)
                    poids=distance+vol
                    parent=[node_actuel[0].pos_x,node_actuel[0].pos_y]
                    position=[x,y]
                    node=Node(position,distance,vol,parent) # on crée le nouveau Node accessible
                    self.liste_ouverte.append([node, poids]) # on l'ajoute avec son poids
                else: # s'il est dans la liste ouverte
                    distance=node_actuel[0].distance+1
                    j=self.index_node_liste_ouverte(x,y) # on cherche sa position
                    if(distance<self.liste_ouverte[j][0].distance):  # si la distance du depart jusqu'à ce node en passant par node actuel est plus courte que la distance déjà enregistrée pour ce node
                        self.liste_ouverte[j][0].parent=[node_actuel[0].pos_x,node_actuel[0].pos_y] # on modifie le node parent et la distance.
                        self.liste_ouverte[j][0].distance=node_actuel[0].distance+1
                        poids=self.liste_ouverte[j][0].distance+self.liste_ouverte[j][0].vol_oiseau
                        self.liste_ouverte[j][1]=poids

    def construire_chemin(self,arrive):
        """Fonction appelée par pathfinding() pour retrouver le chemin le plus court.
        Elle prend en argument le Node d'arrivée."""

        chemin=[self.fin] # on commence par mettre la position finale
        chemin.append(arrive.parent)  # on remonte par les parents
        continuer=1
        i=1

        while(chemin[i][0]!=self.depart.pos_x or chemin[i][1]!=self.depart.pos_y): # tant que l'on est pas revenu au départ
            node=self.get_node_liste_fermee(chemin[i])
            chemin.append(node.parent) # on ajoute le Node parent
            i=i+1

        chemin.reverse() # on retourne le chemin pour l'avoir du départ vers l'arrivée
        del chemin[0] # on supprime la position de depart
        return chemin
```

```
    def trouver_poids_min(self):
        """Fonction servant à trouver le Node de poids minimum de la liste ouverte.
        Elle retourne ce Node."""

        mini=73 # dans le cas le plus défavorable, le chemin fait 72 déplacements
        valeur=0
        node=self.liste_ouverte[0]
        for iterateur in self.liste_ouverte: # pour tout les Nodes de la liste ouverte
            valeur = iterateur[1]
            if(valeur<mini): # on compare le poids du Node au poids minimal trouvé jusque là
                mini=valeur # si le nouveau Node a un poids plus faible il devient le Node au poids mini
                node=iterateur
        node_valeur=[node[0],mini] # le Node trouvé a le poids mini
        return node_valeur

    def calcul_vol_oiseau(self,x):
        """Fonction servant à trouver la distance dans le meilleur des cas jusqu'à l'arrivée.
        Elle prend en argument la position sur x du Node.
        Elle retourne cette distance."""

        abscisse=self.fin-x # le distance sur x restante vaut au mieux la différence d'abscisse entre le Node et la ligne d'arrivée
        return abs(abscisse//2) # on se déplace de 2 cases par 2 cases et on veut un nombre de coups d'où la valeur absolue

    def enlever_node_liste_ouverte(self,node):
        """Fonction servant à enlever un Node de la liste ouverte.
        Elle prend en argument le Node à enlever."""

        i=0 # la recherche débute au premier élément
        for iterateur in self.liste_ouverte:  # pour chaque élément de la liste fermée, un élément contenant un Node et son poids
            if(iterateur[0].pos_x==node.pos_x and iterateur[0].pos_y==node.pos_y): # si le Node de la liste ouverte a la même position que le Node à enlever
                del self.liste_ouverte[i] # c'est le Node recherché, on l'enlève
            i=i+1 # on est passés à l'élément suivant

    def chercher_node_liste_xy(self,liste,x,y):
        """Fonction servant à chercher un Node dans une liste.
        Elle prend en argument cette liste et les coordonnées (abscisse et ordonnée) correspondant au Node à enlever.
        Elle renvoie 1 si le Node est présent dans la liste, 0 sinon."""

        for iterateur in liste: # pour chaque élément de la liste fermée, un élément contenant un Node et son poids
            if(iterateur[0].pos_x==x and iterateur[0].pos_y==y): return 1  # si les positions x et y correspondent, le Node est présent dans la liste
        return 0 # on a parcouru toute la liste sans trouver le Node

    def index_node_liste_ouverte(self,x,y):
        """Fonction servant à chercher la position d'un Node dans la liste ouverte.
        Elle prend en argument cette liste et les coordonnées (abscisse et ordonnée) correspondant au Node à trouver.
        Elle renvoie l'indice de la case du Node s'il est présent dans la liste, -1 sinon."""
```

```
        i=0 # la recherche débute au premier élément
        for iterateur in self.liste_ouverte: # pour chaque élément de la liste fermée, un élément contenant un Node et son poids
            if(iterateur[0].pos_x==x and iterateur[0].pos_y==y): return i # si le Node de la liste ouverte a la bonne position, on renvoie sa position dans la liste
            i=i+1  # on est passés à l'élément suivant
        return -1 # on a parcouru toute la liste sans trouver le Node

    def chercher_node_liste_fermee(self,node):
        """Fonction servant à chercher un Node dans la liste fermée.
        Elle prend en argument le Node recherché.
        Elle renvoie 1 si le Node est présent dans la liste, 0 sinon."""

        for iterateur in self.liste_fermee: # pour chaque élément de la liste fermée, un élément contenant un Node et son poids
            if(iterateur[0].pos_x==node.pos_x and iterateur[0].pos_y==node.pos_y): return 1 # si le Node de la liste ouverte a la bonne position, on a trouvé le Node
        return 0 # on a parcouru toute la liste sans trouver le Node

    def get_node_liste_fermee(self,coordonnee):
        """Fonction servant à chercher un Node dans la liste fermée.
        Elle prend en argument les coordonnées (abscisse et ordonnée) du Node recherché.
        Elle renvoie le Node s'il est présent dans la liste, 0 sinon."""

        node=0 # initialisation d'une variable qui restera à 0 si le Node n'est pas dans la liste fermée
        for iterateur in self.liste_fermee: # pour chaque élément de la liste fermée, un élément contenant un Node et son poids
            if(iterateur[0].pos_x==coordonnee[0] and iterateur[0].pos_y==coordonnee[1]): # si les positions x et y correspondent
                node=iterateur[0] # on a trouvé le Node
                break # continuer la recherche est inutile
        return node

    def fini(self):
        """Fonction servant à déterminer si l'arrivée est atteinte.
        Elle renvoie 1 si elle est atteinte, 0 sinon, et la position y de la case d'arrivée si elle existe, 0 sinon."""
        booleen=0
        y_fin=0
        for iterateur in self.liste_fermee: # pour chaque élément de la liste fermée, un élément contenant un Node et son poids
            if(iterateur[0].pos_x==self.fin): # si le Node se trouve sur la ligne d'arrivée
                booleen=1 # l'arrivée est atteinte
                y_fin=iterateur[0].pos_y
                break # continuer la recherche est inutile
        return booleen,y_fin
#_____________________________________________________________________________

pygame.init() # initialisation du module pygame

fenetre = pygame.display.set_mode((1000, 510)) # ouverture d'une fenêtre pygame de 1000 pixels en largeur pour 510 en hauteur
ecran=Rect(0,0,1000,510)
pygame.display.set_caption("Quoridor") # changement du "titre" de la fenêtre
```

```
couleurFond=(107,130,150) # définition de toutes les couleurs utilisées pour l'interface graphique
couleurPlateau=(107,13,13)
couleurCase=(47,27,12)
couleurBarriere=(255,180,0)
couleurBarriereMobile=(255,255,20)
couleurPion1=(255,244,141)
couleurPion2=(255,150,50)


background=pygame.Surface((1000,510)) # création d'un fond d'écran
background.fill(couleurFond)

plateauAffiche=Rect(40,40,460,460) # définition d'un rectangle qui contiendra le plateau
backgroundPlateau=pygame.Surface((460,460))
backgroundPlateau.fill(couleurPlateau) # il aura un fond d'une couleur qui lui est propre

zone = Rect(520,60,480,340) # définition d'un rectangle qui contiendra les instructions au joueur
backgroundZone=pygame.Surface((480,340))
backgroundZone.fill(couleurFond) # il aura un fond de la même couleur que le fond de la fenêtre

zoneBarrieres=Rect(520,400,480,110) # définition d'un rectangle qui contiendra le nombre de barrières
restantes pour chaque joueur

zoneJoueur=Rect(0,0,500,40) # définition d'un rectangle qui indiquera quel joueur doit jouer
backgroundJoueur=pygame.Surface((500,40))
backgroundJoueur.fill(couleurFond) # il aura un fond de la même couleur que le fond de la fenêtre

font=pygame.font.Font(None, 36) # définition de la police d'écriture

jouer=1
while jouer: # boucle qui servira à jouer plusieurs parties à la suite si le(s) joueur(s) le souhaite(nt)

    fenetre.blit(background,(0,0)) # le fond d'écran occupe toute la fenêtre et efface ce qui aurait pu rester
d'une partie précédente
    fenetre.blit(backgroundPlateau,plateauAffiche) # début du dessin du plateau : application du fond

    for k in range(9):
        for j in range(9):
            pygame.draw.rect(fenetre,couleurCase,[50*k+50,50*j+50,40,40]) # dessin des cases du plateau


    pygame.draw.circle(fenetre,couleurPion1,[270,70],15) # dessin des pions à leurs positions respectives de
départ
    pygame.draw.circle(fenetre,couleurPion2,[270,470],15)

    texte="""Voulez vous :

Jouer à 2 sur le même ordinateur ? (F1)
Jouer contre un IA ? (F2)"""

    changementCommandes(texte) # on demande au joueur de choisir son mode de jeu

    pygame.display.update() # actualisation de l'affichage (avant cela aucun changement n'est visible par
l'utilisateur)
```

```
    choix=0
    while not choix: # on récupère son choix via pygame
        pygame.time.Clock().tick(30) # limitation du nombre de tours de boucle par seconde pour ne pas utiliser tout le CPU
        for event in pygame.event.get(): # pour chaque événement enregistré par pygame
            if event.type == QUIT: # si le joueur veut fermer la fenêtre
                jouer=fin(0,0) # appel de la fonction qui ferme proprement la fenêtre après un affichage
                choix=-1 # sortie de la boucle while
            elif event.type == KEYDOWN: # si l'utilisateur a appuyé sur une touche
                if event.key == K_F1: # si c'est F1
                    choix=1 # son choix est 1
                elif event.key == K_F2:
                    choix=2

    if choix!=-1:
        if choix==1: # si choix vaut 1 cela veut dire que 2 joueurs veulent s'affronter sur la même machine
            joueur='P1' # le joueur 1 commencera
            texteJoueur="Au tour du joueur 1"
            fenetre.blit(font.render(texteJoueur,0,(0,0,0)),(160,10)) # on écrit que c'est à lui de jouer
            vsIA=False # il n'y aura pas d'IA
        else:
            joueur='IA' # l'IA commence
            texteJoueur="Au tour de l'IA"
            fenetre.blit(font.render(texteJoueur,0,(0,0,0)),(160,10))
            vsIA=True

        barrieres="""Nb barrières restantes J1 : 10
Nb barrières restantes J2 : 10"""

        x,y = zoneBarrieres.topleft # on va écrire depuis le coin en haut à gauche de l'emplacement dédié

        for ligne in barrieres.splitlines(): # pour chaque ligne à écrire
            x,y = fenetre.blit(font.render(ligne,0,(0,0,0)),(x,y)).bottomleft # on écrit la ligne et on récupère son coin en bas à gauche pour pouvoir reprendre l'écriture à ce point pour ne pas écrire les lignes les unes sur les autres (pas de chevauchement)

        texteCommandes = """Commandes

Pion :
Appuyer sur p
Choisir la direction à l'aide des flèches.

Barrière :
Appuyer sur b
Positionner à l'aide des flèches.
r pour tourner la barrière;
Entrer pour valider position et sens"""

        changementCommandes(texteCommandes) # le texte contenant les instructions pour l'utilisateur est actualisé via cette fonction (mais elle ne rend pas le changement visible)
        pygame.display.update() # on rend ainsi le changement visible
```

```
        x_P1=0 # le pion 1 est à la ligne 0 colonne 8 (au milieu en haut)
        y_P1=8
        x_P2=16 # le pion 2 à la ligne 16 colonne 8 (au milieu en bas)
        y_P2=8
        plateau=initialisationPlateau(vsIA) # création de la matrice (tableau) représentant le plateau
        pygame.image.save(fenetre,'fenetre.bmp') # on enregistre la fenêtre sous la forme d'un fichier bmp, ce qui permettra de revenir à cet affichage (voir Jouer.barriereMobile() )
        jeu=Jouer(joueur,plateau,x_P1,y_P1,x_P2,y_P2,10,10,fenetre,vsIA) # création de la classe qui va gérer la partie
        gagnant=jeu.toursDeJeu() # appel de la fonction de la classe qui permet au(x) joueur(s) de jouer
        jouer=fin(gagnant,vsIA) # jouer vaudra 1 pour une nouvelle partie (pour rester dans la boucle while), 0 sinon ; suivant le choix de l'utilisateur

pygame.quit() # fermeture de tout ce qui à rapport à pygame, notamment la fenêtre
```