



27 Novembre 2020



Projet IA navigation à voile

[Intelligence Artificielle -1]

GADEAU Juliette - PERRIER Alban

BORDEAUX INP

ENSC 2A – Groupe 1

SOMMAIRE

PRÉSENTATION DU PROJET	2
RÉPARTITION DES TÂCHES	3
MODÉLISATION DU PROBLÈME	4
Stratégie de résolution	4
Modélisation	4
Structure du projet	5
Nos méthodes	6
Saisie et affichage	7
NOTRE HEURISTIQUE	10
RÉSULTATS OBTENUS	12
Cas A	12
Cas B	13
Cas C	14

1) PRÉSENTATION DU PROJET

Ce rapport rend compte de notre travail concernant le projet du module Bases de l'Intelligence Artificielle. Le but de ce projet est de réaliser un programme déterminant le trajet optimal d'un voilier positionné initialement en un point (x_0, y_0) pour rejoindre en un minimum de temps un point d'arrivée (x_f, y_f) . La zone navigable est définie par un carré de 300 km de côté. La détermination du trajet optimal se fait par l'application de la méthode A*. Ce travail se fait en s'appuyant sur trois fonctions et deux classes fournies.

Notre projet est disponible à ce lien github : https://github.com/aperrier004/ENSC_2A_IA/

2) RÉPARTITION DES TÂCHES

Pour réaliser le projet, nous avons travaillé de concert. En effet, lors de nos séances de travail, nous nous sommes rapprochés d'un fonctionnement en mode présentiel. Ainsi, seul l'un de nous deux avait son éditeur de code ouvert avec son écran partagé. De plus, nous avons utilisé GIT pour s'échanger le code facilement. Ceci s'est fait dans le but d'alterner la personne en charge du partage de l'écran, mais également lorsque chacun de notre côté nous travaillions sur le projet pour des avancements mineurs.

Voici notre tableau de synthèse :

Tâche	Alban	Juliette	Date de début	Date de fin
Création du WinForm et de ses éléments de contrôle	X	X	06/11	06/11
Implémentation des fonctions annexes et des classes GenericNode et SearchTree	X	X	06/11	06/11
Création de la classe NodeNavigation	X	X	06/11	06/11
Implémentation des fonctions <i>IsEqual()</i> et <i>GetArcCost()</i>	X	X	06/11	06/11
Gestion des événements relatifs au choix du type de vent	X	X	12/11	12/11
Implémentation des fonctions <i>EndState()</i> , <i>GetListSucc()</i> et <i>VerifCoord()</i>	X	X	12/11	12/11
Implémentation de la fonction <i>tracerSegment()</i>	X		12/11	17/11
Gestion des événements liés aux éléments de contrôle pour récupérer les données	X	X	12/11	17/11
Gestion de l'événement pour afficher les résultats	X	X	15/11	15/11
Gestion des feed-backs et des exceptions d'entrées utilisateurs	X		17/11	17/11
Implémentation de <i>CalculeHCost()</i>	X	X	17/11	20/11
Finalisation du Look&Feel		X	19/11	19/11

3) MODÉLISATION DU PROBLÈME

Stratégie de résolution

L'objectif ici est de trouver le chemin le plus optimal, c'est-à-dire le plus **rapide**, que le bateau doit parcourir entre le point de départ et le point d'arrivée.

Pour cela, l'algorithme A* est utilisé : il se base sur le fonctionnement de l'algorithme de Dijkstra, mais avec l'ajout d'une fonction heuristique. En effet, notre problème peut être décrit par un **graphe** dont les sommets (ou **nœuds**) correspondent aux différents états du bateau et les relations correspondent aux transitions possibles entre ces états.

L'**algorithme de Dijkstra** permet alors de trouver le court chemin entre deux **nœuds** du graphe par une exploration de ce graphe : au fur et à mesure, chaque **nœud** est classé parmi les Ouverts ou les Fermés, et l'on s'arrête lorsque le **nœud** final est placé dans les Fermés. L'ajout d'une fonction heuristique dans l'**algorithme A*** entraîne un changement de stratégie de classement des **nœuds** dans les ouverts, stratégie plus efficace. C'est ce dernier algorithme que nous utilisons pour résoudre notre problème.

Modélisation

La **zone navigable**, qui est un carré de 300 km de côté, est modélisée par une **matrice** de taille 300 par 300. L'affichage de cette zone correspond à une image de taille 300x300, où donc un pixel correspond à une case de la matrice.

La modélisation du problème afin de lui appliquer A* se fait de la manière suivante :

- Un **nœud** du graphe d'état correspond aux coordonnées d'une position du bateau dans la zone navigable, c'est-à-dire aux coordonnées d'une case de la matrice.
- Le **coût d'un chemin** représente le temps mis pour le parcourir par le bateau. Ce temps dépend de la vitesse et de la direction du vent, et est renvoyé par la méthode *GetArcCost()* de la classe NodeNavigation.

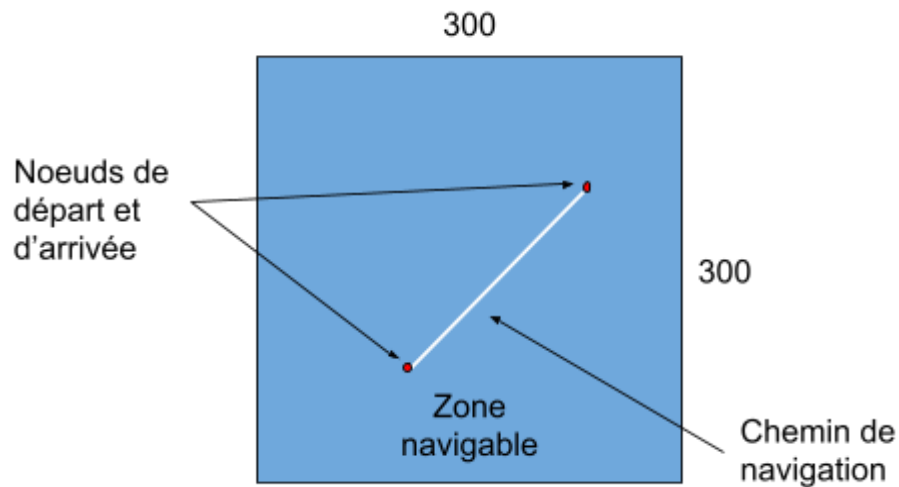


Figure 1 - Schéma modélisant le problème

Structure du projet

Notre projet est une application de type Windows Form, elle est composée des classes suivantes :

- GenericNode (fourni en annexe du sujet du projet) : classe abstraite permettant de résoudre des problèmes avec une structure de noeuds
- SearchTree (fourni en annexe du sujet du projet) : classe contenant notamment l'algorithme A* et permettant de générer une solution
- NodeNavigation : classe héritant de GenericNode, où l'on redéfinit les méthodes permettant de gérer les noeuds adapté à notre contexte et notre solution
- MainForm : classe de type WinForm, elle contient le design de notre fenêtre d'application, ainsi que toutes les gestions d'événements sur ses éléments de contrôle

Nos méthodes

Les méthodes surchargées de la classe GenericNode :

GetArcCost()

Entrée : un objet GenericNode N2

Sortie : un double

Description : appelle la fonction *time_estimation()* permettant de renvoyer un double correspondant au coût d'un chemin et donc le temps mis par le bateau pour aller d'un point A à un point B.

GetListSucc()

Entrée : /

Sortie : une liste d'objets de type GenericNode

Description : vérifie les coordonnées de tous les nœuds voisins et s'ils sont possibles (c'est-à-dire pas en dehors de l'image ou pas déjà fermé/parcouru) les ajoute à une liste de nœuds qui sera renvoyée.

EndState()

Entrée : /

Sortie : un booléen

Description : renvoie True si l'objet de type GenericNode étudié correspond au point d'arrivée du bateau saisi par l'utilisateur.

IsEqual()

Entrée : un objet GenericNode N2

Sortie : un booléen

Description : renvoie True si l'égalité entre deux objets de type GenericNode est vérifiée.

CalculeHCost()

Entrée : /

Sortie : un réel

Description : renvoie le temps estimé entre le nœud courant et le nœud final ([cf. notre fonction heuristique](#)).

Les méthodes que nous avons créée :

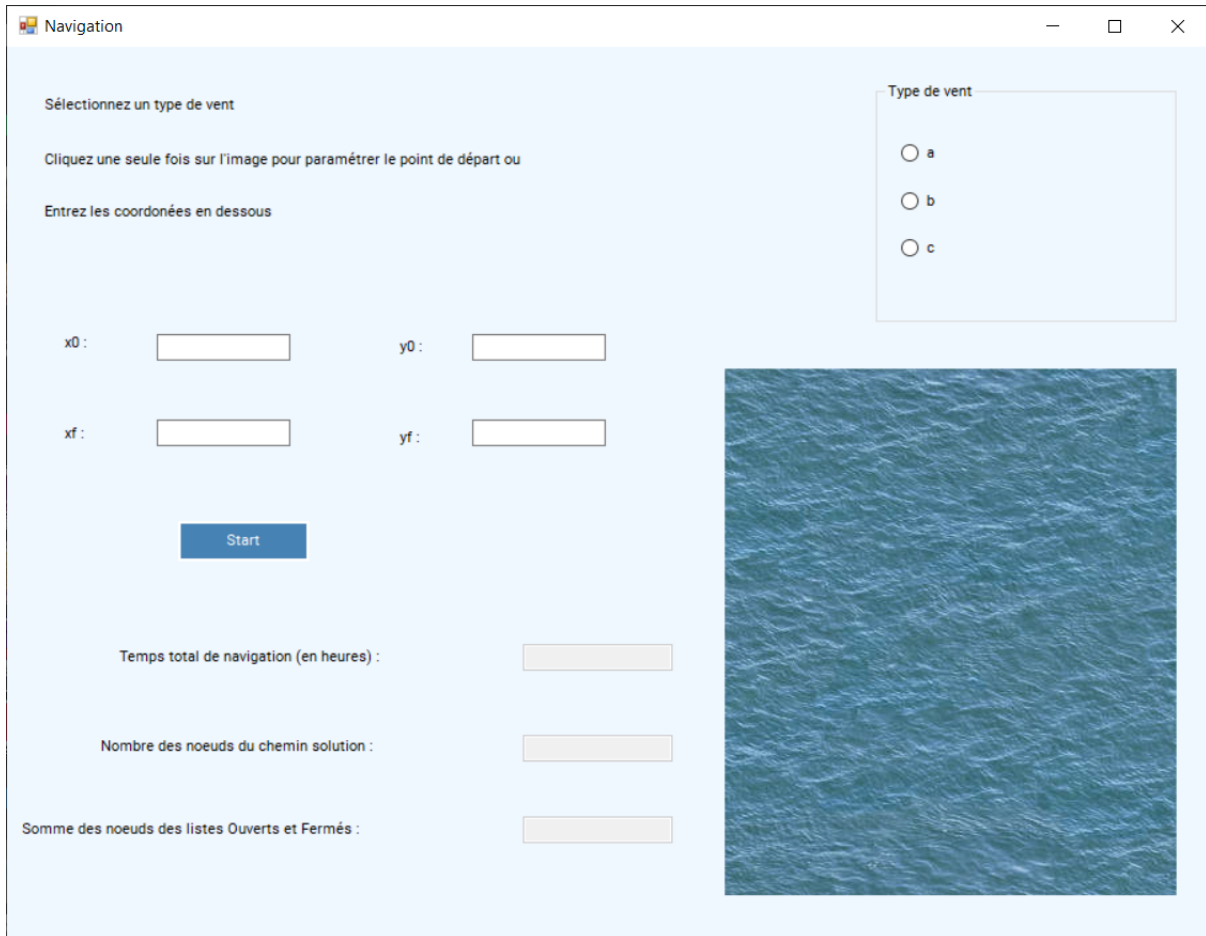
VerifCoord()

Entrée : deux entiers représentants des coordonnées

Sortie : un booléen

Description : renvoie True si les coordonnées étudiées sont contenues dans la zone navigable.

Saisie et affichage



The screenshot shows a window titled "Navigation" with a light blue background. It contains several input fields and a button:

- Text: "Sélectionnez un type de vent"
- Text: "Cliquez une seule fois sur l'image pour paramétrer le point de départ ou"
- Text: "Entrez les coordonnées en dessous"
- Input fields for coordinates: x_0 , y_0 , x_f , and y_f .
- A blue "Start" button.
- Input fields for navigation parameters: "Temps total de navigation (en heures) :", "Nombre des noeuds du chemin solution :", and "Somme des noeuds des listes Ouverts et Fermés :".
- A "Type de vent" section with three radio buttons labeled "a", "b", and "c".
- A large image of a blue sea surface on the right side.

Figure 2 - Écran de l'application à l'ouverture

La saisie des coordonnées des points de départ et d'arrivée du bateau se fait via une interface **Window Forms**. Soit l'utilisateur entre manuellement ces coordonnées, soit il décide de cliquer sur l'image représentant la zone navigable pour choisir les deux points (événement *MouseClicked*). Nous avons laissé ce choix à l'utilisateur afin de pouvoir tester directement les trois scénarios demandés dans l'énoncé. De plus, il doit également sélectionner parmi trois éléments *radioButtons* le type de vent qu'il souhaite.

Une fois ces paramètres entrés, l'utilisateur peut déclencher l'événement *Click* sur le bouton *Start*. En conséquence, le **chemin** le plus rapide qui a été trouvé s'affiche sous forme de segments blanc sur l'image de la zone navigable. Les points de départ et d'arrivée sont représentés par des cercles rouges sur cette même image (seul leur pixel en haut à gauche correspond réellement à la coordonnée exacte). Le **temps total** de navigation en heures (arrondi pour un affichage pertinent), le **nombre de nœuds du chemin solution** ainsi que la **somme des nœuds** des listes des **Ouverts** et des **Fermés** s'affichent sur l'interface. En plus de ces résultats, on affiche un bouton *Restart* qui permet de relancer l'application pour ne pas avoir à la quitter pour tenter un nouveau parcours.

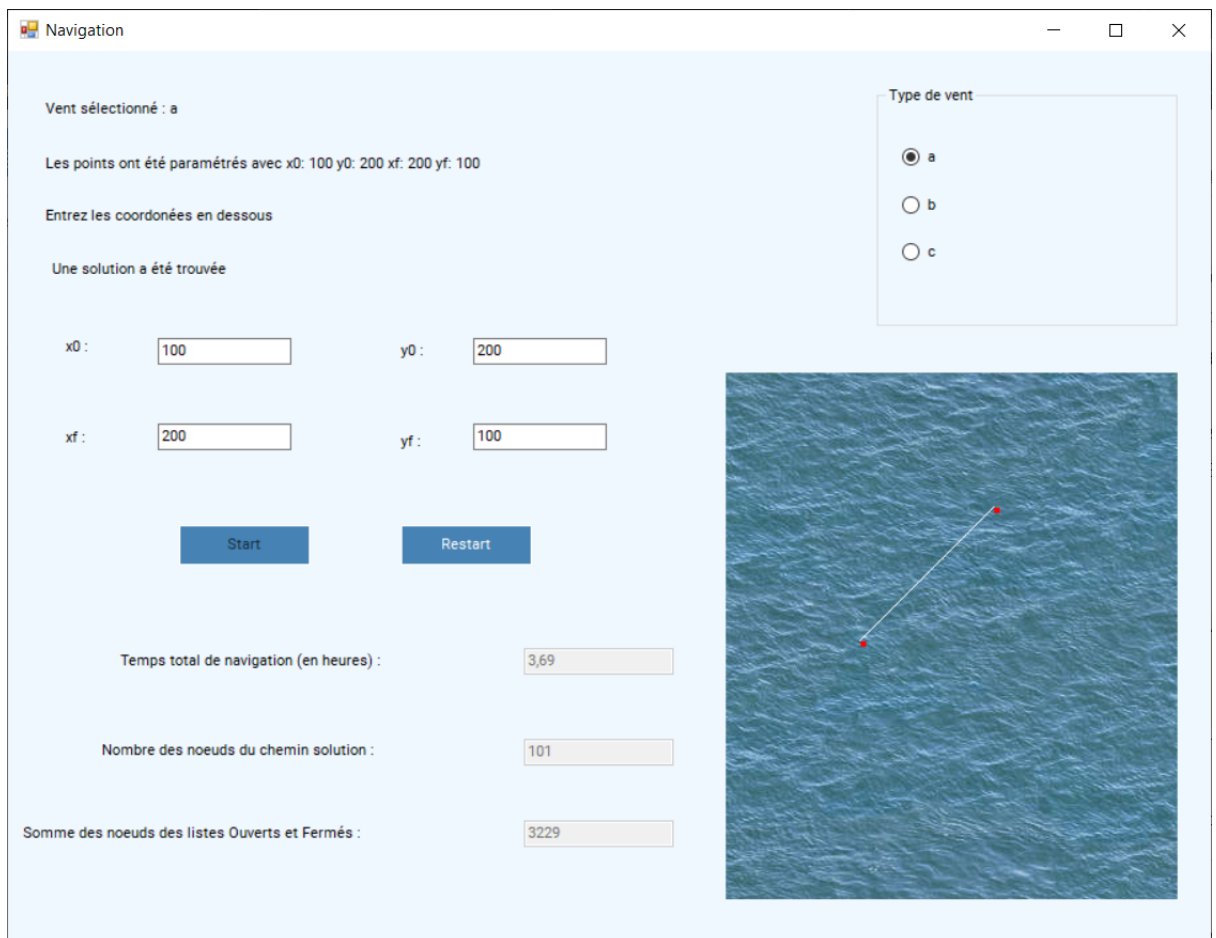
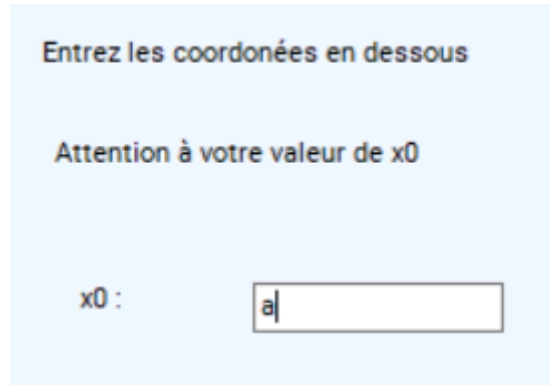


Figure 3 - Écran de l'application une fois que l'on a donné des paramètres et cliqué sur le bouton "Start"

Concernant les différentes saisies de l'utilisateur, nous avons veillé à ce qu'elle ne puisse pas déclencher de comportement inattendu de la part de notre application. En effet, concernant les entrées claviers, nous vérifions qu'il s'agit de nombres cohérents avec ce que l'on attend, et si ce n'est pas le cas, nous affichons un feed-back à l'utilisateur.



Entrez les coordonnées en dessous

Attention à votre valeur de x_0

x_0 :

Figure 4 - Exemple de feed-back lorsque la valeur entrée n'est pas bonne

Nous avons également protégé l'application de clics parasites. Par exemple, nous ne permettons pas à l'utilisateur de cliquer plus que deux fois sur l'image s'il a choisi d'entrer ses coordonnées par le clic. Dans le cas où il choisit de les entrer par clavier, nous désactivons immédiatement la possibilité de sélectionner graphiquement les coordonnées. De manière plus générale, nous désactivons les éléments au fur et à mesure du parcours de l'utilisateur dans l'application. Par exemple, le bouton *Restart* n'apparaît que lorsque la solution a été exécutée et que plus aucune action n'est possible.

4) NOTRE HEURISTIQUE

Notre fonction heuristique *CalculeHCost()* estime le temps de navigation entre le nœud étudié et le nœud final. Ce temps est calculé à partir de la **distance** euclidienne entre ces deux points, **divisée** par la **vitesse maximale** du bateau qui est de **45 km/h**.

En effet, la vitesse maximale du vent est de 50 km/h et dans ce cas-là, d'après le graphique fourni suivant, la vitesse maximale du bateau est de 45 km/h.

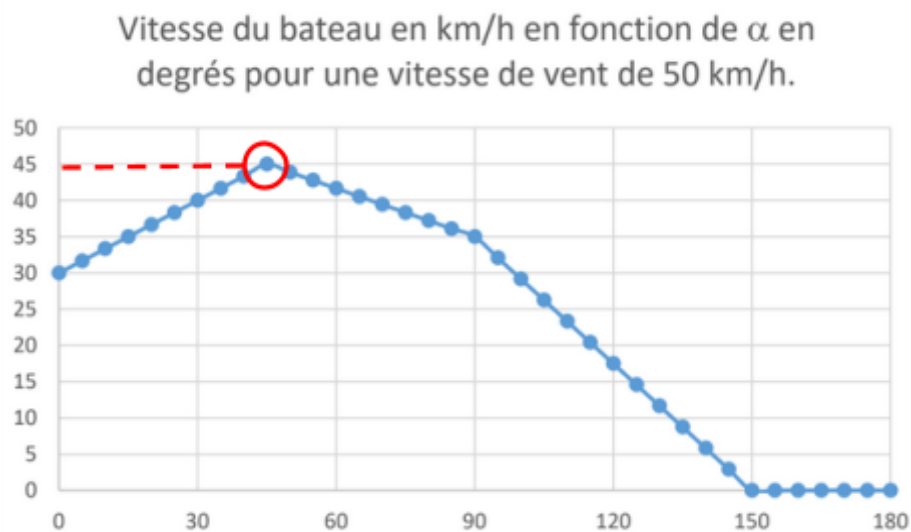


Figure 5 - Graphique représentant la vitesse du bateau en fonction de alpha

Ainsi, nous calculons le coût d'un déplacement sans les contraintes liées au vent. Plus ce coût est faible, plus le nœud est favorable.

En plus de cette fonction, nous avons dû mettre en place une stratégie pour **optimiser le nombre de nœuds ouverts** durant la recherche de solution. En effet, lorsque l'on considérait le cas B, notre application mettait un temps déraisonnable à trouver une solution, notamment à cause du fait que beaucoup trop de nœuds inutiles étaient parcourus. Afin de contrer cela, nous avons décidé d'ajouter une condition pour valider si un nœud devait être ajouté à la liste des successeurs ou non.

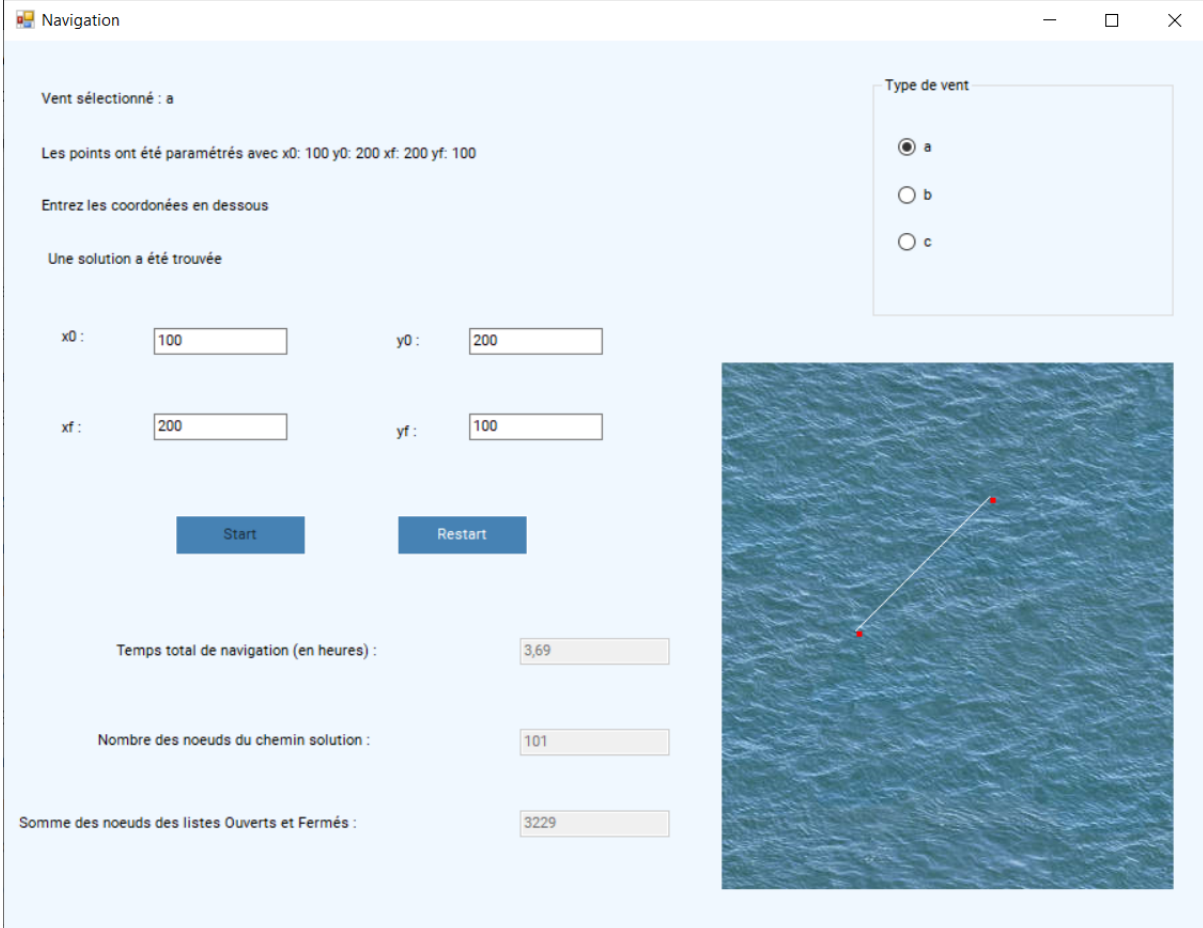
Dans la fonction *GetListSucc()*, nous avons, en plus de la vérification de la coordonnée du nœud, vérifier si la distance du nœud voisin par rapport au point d'arrivée n'était pas supérieure à la distance du nœud courant. Cette méthode nous a alors permis de **réduire drastiquement** le nombre de nœuds parcourus, et en conséquence, l'algorithme s'exécute plus rapidement et nous permet d'afficher un résultat dans un temps raisonnable.

Nous sommes toutefois conscients de ce parti-pris avec notre stratégie pour améliorer *GetListSucc()*. Le fait de ne considérer que les nœuds les plus proches de la solution est une prise de risque, qui nous a cependant été nécessaire pour faire fonctionner le cas B.

5) RÉSULTATS OBTENUS

Cas A

Dans ce cas, le vent est de type a, avec $(x_0, y_0) = (100, 200)$ et $(x_f, y_f) = (200, 100)$.



The screenshot shows a web application titled "Navigation". It displays the following information:

- Vent sélectionné : a
- Les points ont été paramétrés avec $x_0: 100$ $y_0: 200$ $x_f: 200$ $y_f: 100$
- Entrez les coordonnées en dessous
- Une solution a été trouvée
- Inputs for x_0 (100), y_0 (200), x_f (200), and y_f (100).
- Buttons for "Start" and "Restart".
- Results:
 - Temps total de navigation (en heures) : 3,69
 - Nombre des noeuds du chemin solution : 101
 - Somme des noeuds des listes Ouverts et Fermés : 3229
- A "Type de vent" section with radio buttons for a, b, and c, where 'a' is selected.
- A visual representation of the sea with a straight line connecting two red dots, representing the optimal path.

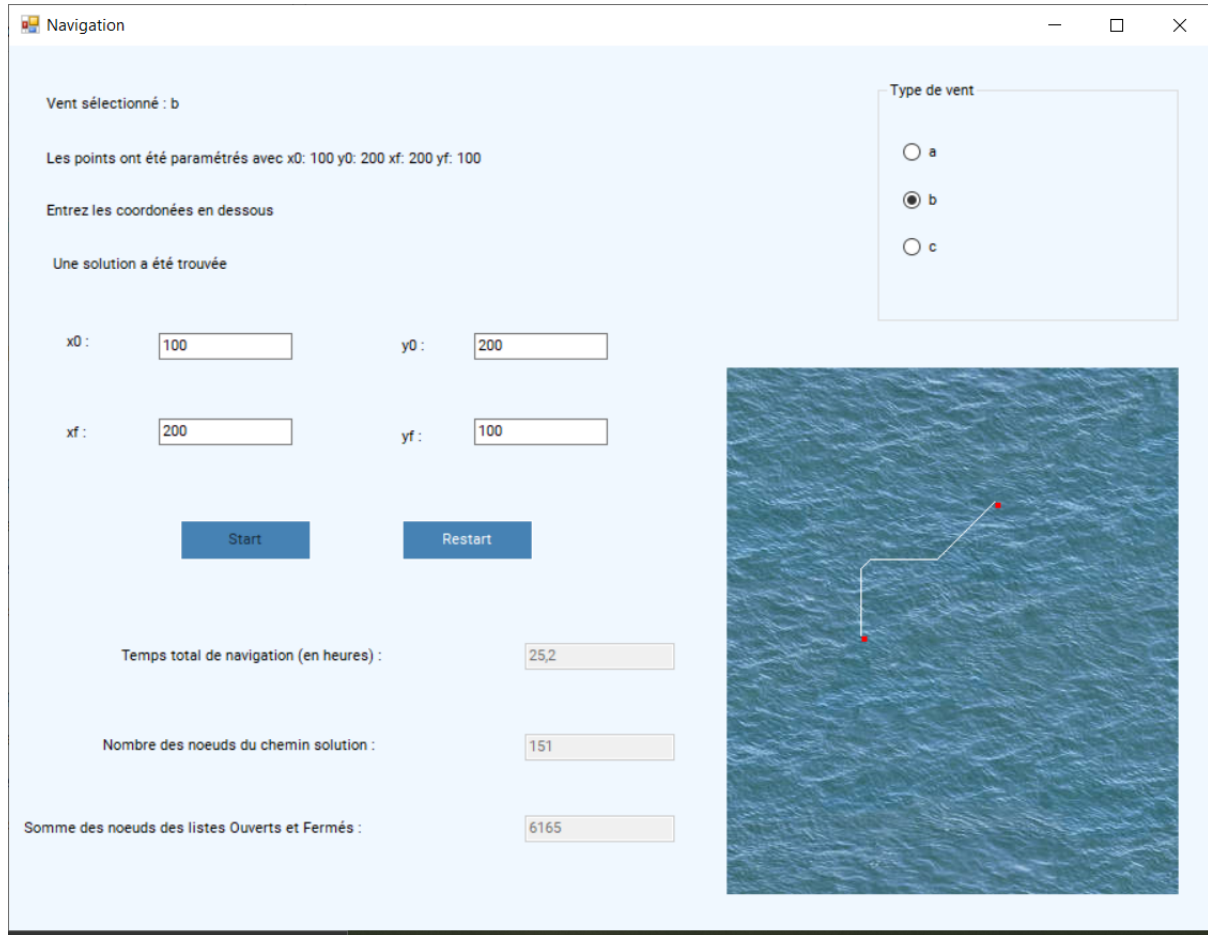
Figure 6 - Écran solution du cas A

On observe donc que le chemin le plus rapide trouvé est composé de 101 nœuds et qu'il forme à priori une ligne droite. 3229 nœuds ont cependant été parcourus pour trouver ce chemin solution.

Le bateau met donc 3.69 heures (environ 3h et 41 minutes) à naviguer entre le point d'arrivée et de départ.

Cas B

Dans ce cas, le vent est de type b, avec $(x_0, y_0) = (100, 200)$ et $(x_f, y_f) = (200, 100)$.



The screenshot shows a web application titled "Navigation". It displays the following information:

- Vent sélectionné : b
- Les points ont été paramétrés avec $x_0: 100$ $y_0: 200$ $x_f: 200$ $y_f: 100$
- Entrez les coordonnées en dessous
- Une solution a été trouvée
- Inputs: $x_0: 100$, $y_0: 200$, $x_f: 200$, $y_f: 100$
- Buttons: Start, Restart
- Temps total de navigation (en heures) : 25,2
- Nombre des noeuds du chemin solution : 151
- Somme des noeuds des listes Ouverts et Fermés : 6165
- A map on the right showing a path from a red dot at (100, 200) to another red dot at (200, 100) over a blue sea background. The path consists of a vertical line segment, a horizontal line segment, and a diagonal line segment.
- A "Type de vent" section with radio buttons for a, b (selected), and c.

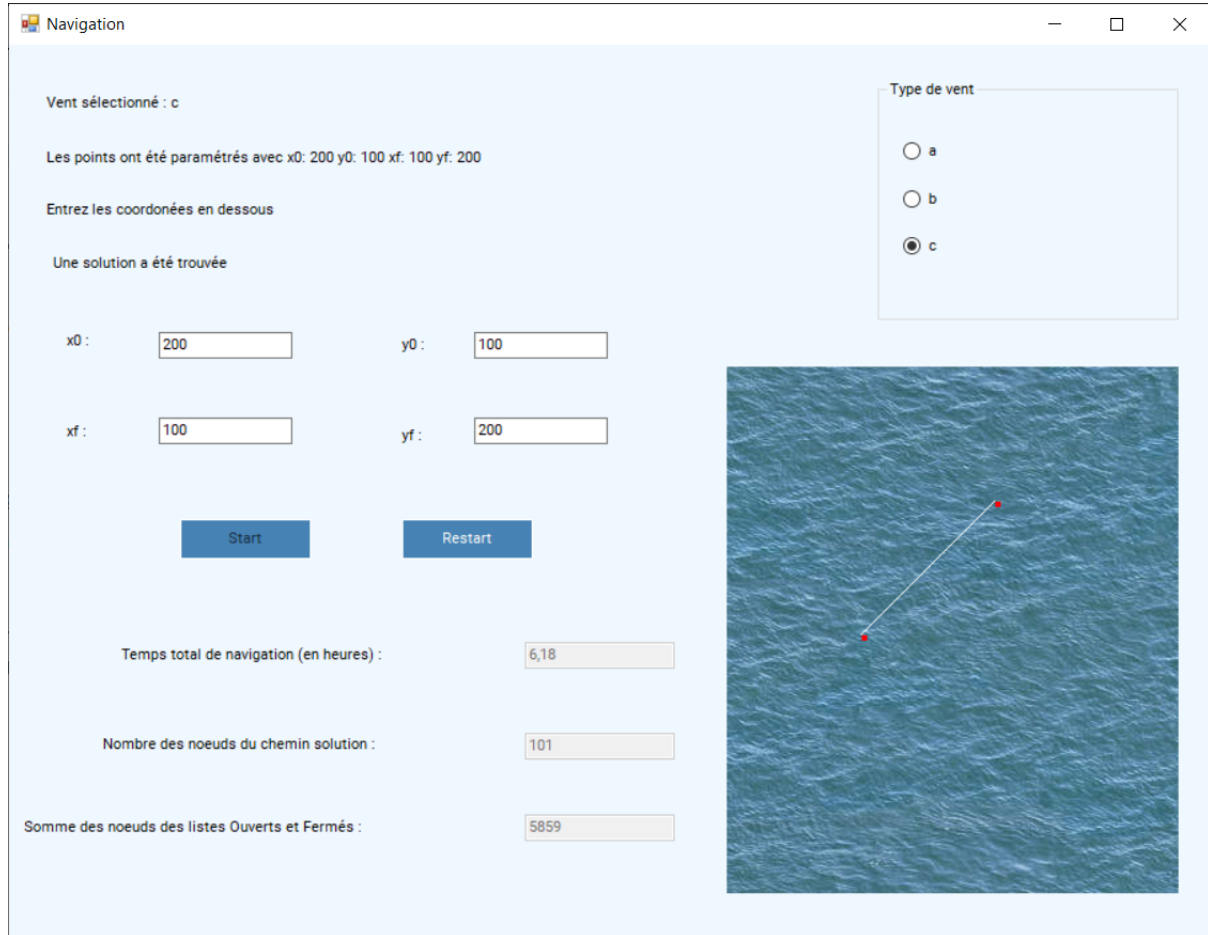
Figure 7 - Écran solution du cas B

On observe donc que le chemin le plus rapide trouvé est composé de 151 nœuds et qu'il forme à priori une première ligne verticale, pour ensuite tourner à droite horizontalement et enfin tracer une diagonale. 6165 nœuds ont cependant été parcourus pour trouver ce chemin solution.

Le bateau met donc 25,2 heures (environ 25h et 12 minutes) à naviguer entre le point d'arrivée et de départ.

Cas C

Dans ce cas, le vent est de type c, avec $(x_0, y_0) = (200, 100)$ et $(x_f, y_f) = (100, 200)$.



The screenshot shows a web application titled "Navigation". It displays the following information:

- Vent sélectionné : c
- Les points ont été paramétrés avec $x_0: 200$ $y_0: 100$ $x_f: 100$ $y_f: 200$
- Entrez les coordonnées en dessous
- Une solution a été trouvée
- Inputs: $x_0: 200$, $y_0: 100$, $x_f: 100$, $y_f: 200$
- Buttons: Start, Restart
- Temps total de navigation (en heures) : 6,18
- Nombre des noeuds du chemin solution : 101
- Somme des noeuds des listes Ouverts et Fermés : 5859
- Type de vent: a, b, c (c is selected)
- A map showing the sea with a straight line path between two red dots representing the start and end points.

Figure 8 - Écran solution du cas C

On observe donc que le chemin le plus rapide trouvé est composé de 101 nœuds et qu'il forme à priori une ligne droite. 5859 nœuds ont cependant été parcourus pour trouver ce chemin solution.

Le bateau met donc 6,18 heures (environ 6h et 11 minutes) à naviguer entre le point d'arrivée et de départ.

On peut remarquer que le cas C est l'inverse du cas A en termes de points d'arrivée et de départ. Or, le nombre de nœuds ouverts et fermés est multiplié par 1.8, et le temps total de navigation par 1.9. Cela démontre l'impact du vent sur le temps de navigation d'un bateau.