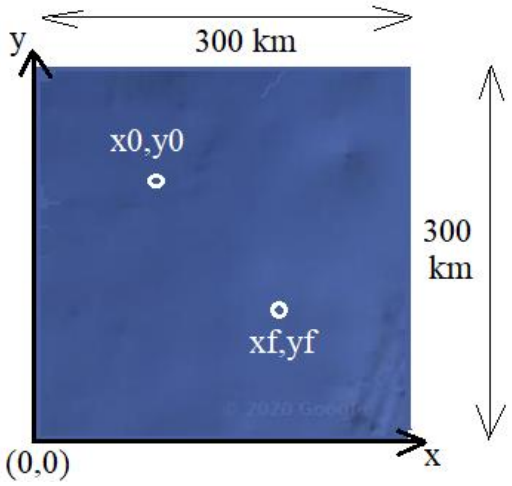
ENSC 2ème année

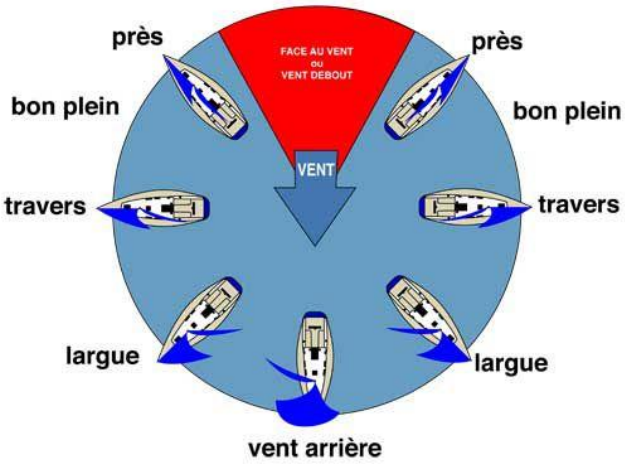
Projet du module Bases de l'Intelligence Artificielle

Par groupe de 2 ou 3 élèves.
L'objectif de ce projet est de réaliser un programme qui va déterminer le trajet optimal d'un voilier positionné initialement en un point (x0, y0) pour rejoindre en un minimum de temps un point d'arrivée (xf, yf). La zone navigable est définie par un carré de 300 km de côté.



On dispose pour cela d'une fonction permettant d'estimer un temps élémentaire de déplacement entre deux points proches. Cette fonction prend en paramètres 2 points, un point de départ $P_1(x_1,y_1)$ et un point d'arrivée $P_2(x_2,y_2)$ et elle est basée sur le modèle suivant :

- Si les 2 points sont éloignés de plus de 10 km, la fonction renvoie une constante égale à 1000000 pour signifier qu'il y a un appel incorrect à la fonction. Le vent dépend en effet de la position du bateau, il n'est donc pas possible d'appeler cette fonction pour des distances importantes.
- Sinon, cela dépend à la fois de la direction d du vent et de sa vitesse v_v au point milieu entre p1 et p2, et de la position des deux points. Le modèle est décrit ci-dessous :



Un premier calcul donne la vitesse du bateau v_b. Soit α la valeur absolue de l'angle entre la direction du vent et la direction d'avancement du bateau, définie par le vecteur $\overrightarrow{P_1P_2}$

- Entre vent arrière et largue, soit pour α entre 0 et 45°

$$v_b = \left(0.6 + \frac{0.3\alpha}{45}\right) v_v$$

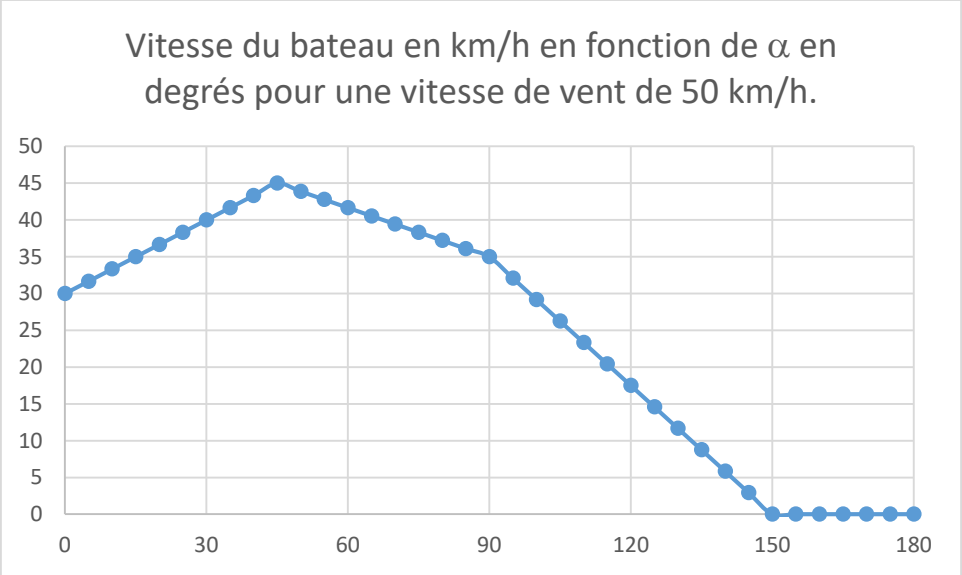
- Entre largue et travers, soit pour α entre 45 et 90°

$$v_b = \left(0.9 - \frac{0.2(\alpha - 45)}{45}\right) v_v$$

- Entre travers et près et jusqu'à 150°, soit pour α entre 90° et 150°

$$v_b = 0.7\left(1 - \frac{\alpha - 90}{60}\right) v_v$$

Pour 150 et au-delà de 150°, le bateau est face au vent et ne peut avancer, sa vitesse est nulle.
Voici le profil de vitesse obtenu en fonction de la différence angulaire α.



Le temps est calculé ensuite directement :

Si la vitesse est nulle, on retourne par convention t=1000000

Sinon $t = \frac{\sqrt{(x1-x2)^2+(y1-y2)^2}}{v_b}$

La fonction qui calcule t, appelée time_estimation, a été écrite et vous est fournie en annexe 1 en langage C#.
L'objectif est pour vous d'écrire un programme qui va exploiter l'A* pour trouver le chemin le plus rapide pour se rendre du point de départ jusqu'à l'arrivée dans les 3 cas suivants :

a) (x0,y0) = (100, 200) ; (xf,yf) = (200,100) et le vent est constant à 50km/h et souffle dans la direction 30° (vers le nord-est).
b) (x0,y0) = (100, 200) ; (xf,yf) = (200,100).
   Si y>150, le vent est constant à 50km/h et souffle dans la direction 180° (vers l'ouest).
   Si y<=150, le vent est constant à 20 km/h et souffle dans la direction 90° (vers le nord)
c) (x0,y0) = (200, 100) ; (xf,yf) = (100,200).
   Si y>150, le vent est constant à 50km/h et souffle dans la direction 170° (vers l'ouest).
   Si y<=150, le vent est constant à 20 km/h et souffle dans la direction 65° (vers le nord-est)

Les fonctions d'estimation de la vitesse et direction du vent sont données en annexe 2.
Les valeurs de (x0,y0) (xf,yf) doivent être demandées à l'utilisateur, ainsi que le type de vent a, b, ou c.

Démarche générale :

Pour minimiser vos efforts, exploitez les sources C# permettant d'appliquer l'A* de manière générique. Dans ce cas, reprenez sans les modifier les classes ClassGraph et GenericNode et prenez exemple sur une des applications fournies pour les exploiter. Il ne reste qu'à écrire une classe héritée

de GenericNode en surchargeant les fonctions GetArcCost, GetListSucc, EndState, IsEqual et éventuellement celle qui calcule l'heuristique.

Pour GetArcCost, il faut simplement appeler time_estimation avec les bons paramètres.

En ce qui concerne GetListSucc, vous pouvez définir un voisinage de manière dynamique. Par exemple, pour un quadrillage carré, les successeurs d'un points (x,y) peuvent être les 8 voisins (x-1,y-1), (x-1,y), (x-1,y+1), (x,y-1), (x,y+1), (x+1,y-1), (x+1,y) et (x+1,y+1). On peut aussi étendre les voisinages plus loin, ou adopter un pavage différent, ou encore choisir un voisinage plus petit (0,1 km par exemple). Attention, selon votre modèle de voisinage, le point (xf,yf) n'est peut-être jamais atteignable comme point voisin. Une solution est de tester la distance de (xf,yf) et de l'ajouter à la liste des successeurs s'il est assez proche.

Pour éviter que la recherche du meilleur temps ne prenne trop de temps, notamment si vous essayez des voisinages denses, pensez à exploiter des heuristiques pertinents qui vont favoriser les chemins les plus appropriés.

Vous pouvez cependant programmer votre propre A* en exploitant un autre langage informatique.

A rendre pour le 27/11/2020 :

- Un rapport dans lequel vous expliquez la répartition des tâches sous forme d'un tableau de synthèse (trop de déséquilibre implique une note différente pour chaque élève), votre modélisation du problème, éventuellement vos heuristiques. Enfin, décrivez les résultats obtenus, notamment :
    - Pour chaque cas a, b et c, le chemin le plus rapide qui a été trouvé (affichage sous forme de segments), ainsi que le temps total de navigation.
    - Le nombre de nœuds du chemin solution, ainsi que la somme des nœuds des listes Ouverts et Fermés.
- Les sources de votre projet.

Annexe 1 : fonction qui calcule le temps de navigation entre 2 points proches

```
public double time_estimation (double x1, double y1, double x2, double y2)
    {
      double distance = Math.Sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
      if (distance > 10) return 1000000;

      double windspeed = get_wind_speed((x1 + x2) / 2.0, (y1 + y2) / 2.0);
      double winddirection = get_wind_direction((x1 + x2) / 2.0, (y1 + y2) / 2.0);

      double boatspeed;
      double boatdirection = Math.Atan2(y2 - y1, x2 - x1)*180/Math.PI;
      // On ramène entre 0 et 360
      if (boatdirection < 0) boatdirection = boatdirection + 360;
      // calcul de la différence angulaire
      double alpha = Math.Abs(boatdirection - winddirection);
      // On se ramène à une différence entre 0 et 180 :
      if (alpha > 180) alpha = 360 - alpha;

      if (alpha <= 45)
      {
        /* (0.6 + 0.3α / 45) v_v */
        boatspeed = (0.6 + 0.3 * alpha / 45) * windspeed;
      }
      else if (alpha <= 90)
      {
        /*v_b=(0.9-0.2(α-45)/45) v_v */
        boatspeed = (0.9 - 0.2 * (alpha - 45) / 45) * windspeed;
      }
      else if (alpha < 150)
      {
        /* v_b=0.7(1-(α-90)/60) v_v  */
        boatspeed = 0.7 * (1 - (alpha - 90) / 60) * windspeed;
      }
      else
        return 1000000;

      // estimation du temps de navigation entre p1 et p2
      return (distance / boatspeed);

    }
```

Annexe 2 : fonctions pour la vitesse et la direction du vent

```
public char cas = ‘a’;      // à modifier en ‘b’ ou ‘c’ selon le choix de l’utilisateur

public double get_wind_speed ( double x, double y)
      {
         if (cas == 'a')
            return 50;
         else if (cas == 'b')
            if (y > 150)
               return 50;
            else return 20;
         else if (y > 150)
            return 50;
            else return 20;
      }

public double get_wind_direction(double x, double y)
      {
         if (cas == 'a')
            return 30;
         else if (cas == 'b')
            if (y > 150)
               return 180;
            else return 90;
         else if (y > 150)
            return 170;
            else return 65;
      }
```

Annexe 3 : Utilisation des graphiques

Dans un premier temps, vous pouvez insérez sur la fiche une PictureBox et y mettre l'image png de votre choix de taille 300x300.
Ensuite, pour afficher un segment, vous pouvez appeler le code suivant :

```
// soient x1, y1, x2, y2 des double utilisés pour définir les 2 extrémités d'un segment.
Pen penwhite = new Pen(Color.White); // d'autres couleurs sont disponibles
Graphics      g = pictureBox1.CreateGraphics();
g.DrawLine(penwhite,   new Point((int)x1, pictureBox1.Height-(int)y1),
                                new Point((int)x2, pictureBox1.Height-(int)y2));
```