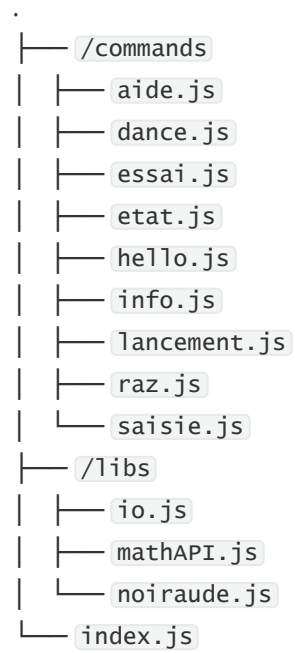


# Annexe - Code source

---

## Arborescence



```

/**
 * @file index.js
 * @requires discord.js
 * @fileoverview Racine du robot discord
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 * @author Julien Mocquet <email>
 */

/** requisition des modules nécessaires */
const Discord = require('discord.js')
var config = require('./config.json');
var fs = require("fs")

/** Initialisation du robot a l'aide du constructeur */
bot = new Discord.Client();

/** mise en place d'affichage automatique des messages d'erreur */
bot.on("error", (e) => console.error(e));
bot.on("warn", (e) => console.warn(e));
//bot.on("debug", (e) => console.info(e));

/** Association du robot avec son jeton de connexion a l'API discord */
bot.login(config.tokenBot);

/** Création d'une collection de commandes du robot */
bot.commands = new Discord.Collection();

/** actions effectués après l'initialisation du root */
bot.once('ready', () => {
  console.info(`${bot.user.tag} launched`);
  bot.user.setActivity("`!aide` pour afficher les commandes");
})

/**
 * importation de chaque module de commande présent dans `./commands/`
 * a la collection de commandes du bots
 */
const commandFiles = fs.readdirSync('./commands').filter(
  file => file.endsWith('.js')
);

for (const file of commandFiles){
  const command = require(`./commands/${file}`);
  bot.commands.set(command.name, command);
}

/** evenement asynchrone effectué quand un message est envoyé dans un salon */
bot.on('message', message => {
  /** ignore tous les messages au type `DEFAULT` et ceux émis par le robot */
  if (message.type !== 'DEFAULT' || message.author.bot){
    return;
  }

  /** Parsage du message en arguments */
  const args = message.content.split(/ +/)

```

```

/**
 * @const nom de la commande */
const commandName = args.shift().toLowerCase()

/** ignore les messages sans préfixe */
if (!commandName.startsWith(config.prefix)){
    return;
}

/** recherche de la commande parmi la collection */
const command = bot.commands.get(commandName.slice(config.prefix.length))

/** en cas de commande non trouvée */
if(!command){
    message.reply(
        "Je ne sais pas faire ça, je vais demander de l'aide à mon médecin"
    );
    return;
}

/** Tentative d'exécution de la commande */
try{
    command.execute(message, args)
}
catch (error){
    console.error(error);
    message.reply("Oops je crois que quelque chose c'est mal passé");
}
})

```

```

/**
 * @file io.js
 * @fileoverview Librairie de formatage entrée/sortie
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

const mathAPI = require('../libs/mathAPI.js');

/** @const saisieFilterRegex - expression régulière pour caractères illégaux */
const saisieFilterRegex = new RegExp(/[a-zA-Z;]+/g);

/**
 * Formatte le json fourni en entrée
 * @param {Object} json a formater
 */
function jsonPrettifier(input){
    input=JSON.stringify(input,null,"\t");
    return input;
}

/**
 * valide ou non la chaine de `!saisie` d'entrée à l'aide de regex
 * @param {String} input Chaine de caractères a analyser
 * @returns true,false
 */
function saisieValidation(input){
    if(input.match(saisieFilterRegex)){
        return false;
    }
    else
    {
        return true;
    }
}

/**
 * Convertit la chaine de caractères formatée en Objet d'ensemble de points
 * @function saisieParser
 * @param {String} input
 * @returns {Object} group Groupe de points
 */
function saisieParser(input){
    let group = input.split(":");
    let tempCoords = [];
    for (const [i, value] of group.entries()) {
        tempCoords = group[i].split(",");
        group[i]=mathAPI.pointConstructor(
            parseFloat(tempCoords[0]),parseFloat(tempCoords[1])
        );
    }
    return group;
}

module.exports = {
    jsonPrettifier,

```

```
saisieValidation,  
saisieParser  
};
```

```

/**
 * @file mathAPI.js
 * @fileoverview Librairie de calcul mathématique
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/**
 * Retourne un objet de point selon les coordonnées
 * fournies en entrée
 * @param {Number} coordX
 * @param {Number} coordY
 * @returns {Object} point
 */
function pointConstructor(coordX, coordY){
    let point = {
        "x": coordX,
        "y": coordY
    }
    return point;
}

/**
 * Retourne la norme du vecteur fourni en entrée
 * @param {number} vecX Cordonnees x du vecteur
 * @param {number} vecY Cordonnees y du vecteur
 * @returns {number} Norme obtenue
 */
function getVectorNorme(vecteur){
    return Math.sqrt(Math.pow(vecteur.x,2)+Math.pow(vecteur.y,2));
}

/**
 * Retourne les coordonnées du vecteur d'entree
 * @param {Vector} input vecteur d'entree
 * @returns {Object} outputVector coordonnées du vecteur
 */
function getVectorCoords(input){
    let outputVector = {x:undefined,y:undefined}
    outputVector.x = input.xb - input.xa;
    outputVector.y = input.yb - input.ya;
    return outputVector;
}

/**
 * obtenir le produit scalaire de deux vecteur d'entree
 * @param {vector} u vecteur
 * @param {vector} v vecteur
 * @returns {number} produit scalaire de sortie
 */
function getProduitScalaire(u,v){
    let tempX = u.x * v.x;
    let tempY = u.y * v.y;
    return tempX+tempY;
}

```

```

/**
 * calculer l'angle entre les segments de droite
 * @param {Segment} vecA
 * @param {Segment} vecB
 * @returns {number} produit scalaire de sortie
 */
function getAngleSegment(vecA,vecB){

    let tempA = getProduitScalaire(vecA,vecB);
    let tempB = getVectorNorme(vecA) * getVectorNorme(vecB)
    let result = Math.acos(tempA/tempB);

    /** Récupération du déterminant et application de son signe a `result` */
    let signe = getDeterminant(vecA,vecB)
    if(signe<0)
    {
        return -result;
    }
    else
    {
        return result
    }
}

/**
 * retourne vrai si G appartient à P et faux sinon
 * @param {polygone} P polygone
 * @param {point} G point
 * @returns {boolean} appartenance
 */
function getAppartenancePointPolygone ( polygone,point){
    let tempPoly = polygone;
    tempPoly.push(tempPoly[0]);
    let result=0;
    let tempVecA;
    let tempVecB;
    for(let i=0;i<polygone.length-1;i++)
    {

        tempVecA=segmentConstructor(point,polygone[i]);

        tempVecB=segmentConstructor(point,polygone[i+1]);
        tempVecA= getVectorCoords(tempVecA);
        tempVecB= getVectorCoords(tempVecB);
        result+=getAngleSegment(tempVecA,tempVecB);
    }
    // si somme des angles==0, point externe au polygone (sinon interne)
    if(result==0)
    {
        return false;
    }
    else
    {
        return true;
    }
}

```

```

/**
 * retourne un segment a partir de deux points fournis en entrée
 * @param {Object} pointA
 * @param {Object} pointB
 */
function segmentConstructor(pointA,pointB){
    let output={xa:pointA.x, ya:pointA.y, xb:pointB.x, yb:pointB.y}
    return output;
}

/**
 * calculer l'aire du polygone fourni en entrée
 * @param {Array} polygone
 * @returns {Number} result Aire obtenue
 */
function getAirePolygone(polygone){
    let result=0;
    let tempPoly = polygone;
    tempPoly.push(polygone[0]);
    for(let i=0;i<tempPoly.length-1;i++){
        tempA = tempPoly[i].x * tempPoly[i+1].y;
        tempB = tempPoly[i+1].x * tempPoly[i].y;
        result += tempA-tempB;
    }
    return result*(0.5);
}

/**
 * Retourne la valeur absolue de la variable donnée en entrée
 * NOTE: même principe d'utilisation que `Math.abs()`...
 * @param {Number} valeur
 * @returns {Number} Valeur absolue
 */
function getValAbsolue(valeur){
    if(valeur<0)
    {
        return -valeur;
    }
    else if(valeur>0 || valeur==0)
    {
        return valeur;
    }
}

/**
 * Calcul du centre de gravité d'un polygone
 * @param {Object} polygone
 * @param {Number} aire
 */
function getCentreGravite(polygone,aire){
    let centreGrav = {};
    centreGrav.x = getAbscisseGravite(polygone,aire);
    centreGrav.y = getOrdonneeGravite(polygone,aire);
    return centreGrav;
}

/**
 * Donne l'abscisse du centre de gravité du polygone

```



```

* @param {Object} polygone
* @param {Number} aire
* @returns {Number} result abscisse du centre de gravité du polygone
*/
function getAbscisseGravite(polygone, aire){
    let result=0;
    let tempPoly=polygone;
    let tempA;
    let tempB;
    let tempAdd;
    tempPoly.push(polygone[0]);

    for(let i=0;i<tempPoly.length-1;i++){
        tempA = tempPoly[i].x * tempPoly[i+1].y;
        tempB = tempPoly[i+1].x * tempPoly[i].y;
        tempAdd = tempPoly[i].x + tempPoly[i+1].x;
        tempC = tempA-tempB;
        result+=(tempAdd)*(tempA-tempB);
    }
    result = result*(1/(6*aire));
    return result;
}

/**
/*
* Obtention de l'abscisse du centre de gravité
* @param {Object} polygone
* @param {Number} aire
* @returns {Number} result abscisse centre de gravité
*/
function getOrdonneeGravite(polygone, aire){
    let result=0;
    let tempPoly=polygone;
    let tempA;
    let tempB;
    let tempAdd;
    tempPoly.push(polygone[0]);

    for(let i=0;i<tempPoly.length-1;i++){
        tempA = tempPoly[i].x * tempPoly[i+1].y;
        tempB = tempPoly[i+1].x * tempPoly[i].y;
        tempAdd = tempPoly[i].y + tempPoly[i+1].y;
        tempC = tempA-tempB;
        result+=(tempAdd)*(tempA-tempB);
    }
    result = result*(1/(6*aire));
    return result;
}

/**
* Obtenir le produit scalaire de deux vecteur d'entree
* @param {Object} vecA premier vecteur d'entrée
* @param {Object} vecB deuxième vecteur d'entrée
* @returns {Number} déterminant des deux vecteurs
* | x.a | y.a | = x.a*y.b - y.a*x.b
* |-----|-----|
* | x.b | y.b |
*/

```

```
function getDeterminant(vecA,vecB){
    let tempX = vecA.x * vecB.y;
    let tempY = vecA.y * vecB.x;
    let result = tempX-tempY;
    return result;
}

/** Exportation des commandes du module */
module.exports ={
    getAirePolygone,
    getCentreGravite,
    getAppartenancePointPolygone,
    getVectorNorme,
    getVectorCoords,
    getProduitScalaire,
    getAngleSegment,
    getAbscisseGravite,
    getOrdonneeGravite,
    pointConstructor,
    getDeterminant,
    getValAbsolue
}
```

```

/**
 * @file noiraude.js
 * @fileoverview Informations stockées sur le pré de la noiraude
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/**
 * {Array} myObj
 * {number} myObj.a
 * {string} myObj.b
 */

/**
 * @type {Array} enclos
 * Tableau composé d'objets de piquets ayant chacun des coordonnées `x` et `y`
 */
var enclos = [];

/**
 * @type {Number} maxPiquets
 * Nombre maximal de piquets autorisés
 */
var maxPiquets = 50;

/** Retourne l'
 * @function getEnclos
 * @returns {Array} enclos
 */
function getEnclos(input){
    return enclos;
}

/** Assigne a l'enclos, le tableau fourni en entrée
 * @function setEnclos
 * @param {Array} value
 * @returns {Array} enclos Enclos modifié
 */
function setEnclos(value){
    enclos=value;
    return enclos;
}

/** Assigne a un emplacement précis de l'enclos, l'objet piquet fourni en entrée

 * @function setEnclosIndex
 * @param {Number} index Index visé
 * @returns {Array} enclos Enclos modifié
 */
function setEnclosIndex(index,value){
    enclos[index]=value;
    return enclos;
}

/** Ajoute a la fin de l'enclos le piquet fourni en entrée
 * @function pushEnclos

```

```

    * @param {Object} value - Objet de piquet
    * @returns {Array} enclos Enclos modifié
    */
    function pushEnclos(value){
        enclos.push(value);
        return enclos;
    }

    module.exports = {
        getEnclos,
        setEnclos,
        setEnclosIndex,
        pushEnclos,
        enclos,
        maxPiquets
    };

```

aide.js

```

/**
 * @file aide.js
 * @usage `!aide`
 * @fileoverview Commande d'affichage des commandes disponibles
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/**
 * Chaîne de caractères brute d'aide afficher (retours a la ligne inclus)
 * @type {String}
 * @const
 */
const aideMessage = `
(réalisé par Théo Gicquel & Julien mocquet)
** Commandes de base **
\!aide\`\t Affichage de la liste de toutes les commandes
\!essai\`\t Charger un enclos d'un jeu d'essai préconfiguré
\!etat\`\t afficher les piquets de l'enclos
\!saisie\`\t saisir un ensemble de piquets de l'enclos simultanément

\!lancement\`\t déterminer si la vache se trouve dans le pré

** Informations & Cosmétique **
\!info\`\t voir les informations sur ce robot
\!hello\`\t dire bonjour
\!dance\`\t ???
`;

/** Exportation du module local en commande appelée via son attribut `name` */
module.exports = {
    execute(message, args){
        message.channel.send(aideMessage);
    },
    name: "aide",
};

```

dance.js

```
/**
 * @file dance.js
 * @usage `!dance`
 * @fileoverview Commande affichant d'un gif animé
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/**
 * Chaîne de caractères de l'url du gif
 * @type {String}
 * @const
 */
var danceUrl = "https://tenor.com/view/polish-dancing-cow-dancing-cow-polish-gif-18921101";

/** Exportation du module local en commande appelée via son attribut `name` */
module.exports = {
  execute(message, args){
    message.channel.send(danceUrl);
  },
  name: "dance",
};
```

```

/**
 * @file essai.js
 * @usage `!essai (1-4)`
 * @fileoverview Commande de remplissage de l'enclos avec les jeux de tests
 * prédéterminés
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/** requisition des modules nécessaires */
let noiraude = require('../libs/noiraude');
let io = require('../libs/io');

/** définition des variables et constantes nécessaires */
let jeu;
let aideEssai=`Veuillez saisir une préconfiguration d'enclos parmi les
suivantes avec `!essai <numéro>` `
let centreAttendu;
let presenceAttendue;
let aireAttendue;

/**
 * Message de présentation des jeux d'essais disponibles
 * @type {String}
 * @const
 */
const listeEssai=`
  \`1\` (-1 , 1) (-1 , -1) (1 , -1) (1 , 1) \n
  \`2\` (-16.6 , -20.1 ) (-12.6 , -18.6) (-11.6 , -16.6) (-15.1 , -15.1 ) \n
  \`3\` (-1.1 , -1.5 ) (2.1 , 3.012 ) (5.6 , -1.21) (1.97 , 4.07) \n
  \`4\` :mouse:\n`

module.exports = {
  execute(message,args){
    if(!args.length)
    {
      message.channel.send(aideEssai);
      return message.channel.send(listeEssai);
    }
    else
    {
      /** Récupération des infos a afficher et données a intégrer */
      switch (parseInt(args[0])){
        case 1:
          jeu="-1,1:-1,-1:1,-1:1,1";
          aireAttendue="4";
          centreAttendu="(0,0)";
          presenceAttendue="Intérieur";
          break;
        case 2:
          jeu="-16.6,-20.1:-12.6,-18.6:-11.6,-16.6:-15.1,-15.1";
          aireAttendue="13.125";
          centreAttendu="(-14.226, -17.555)";
          presenceAttendue="Intérieur";
          break;

```

```

        case 3:
            jeu="-1.1,-1.5:2.1,3.012:5.6,-1.21:1.97,4.07";
            aireAttendue="3.563";
            centreAttendu="(1.978, 1.903)";
            presenceAttendue="Extérieur";
            break;
        case 4:

            jeu="26,0:8,10:10,11:6,15:3,22:1,27:3,29:12,25:17,20:18,22:24,20:33,18:39,8:44,
0:44,-8:44,-18:39,-20:33,-22:24,-20:18,-25:17,-29:12,-27:3,-22:1,-15,3:-11,6:-10
,10"

            aireAttendue="3.563";
            centreAttendu="1.979";
            presenceAttendue="1.904";
            break;

        default:
            return message.channel.send("erreur de saisie");
    }
    message.channel.send(jeu)
    message.channel.send("intégration de l'enclos...");
    noiraude.enclos=io.saisieParser(jeu);
    message.channel.send("aire attendue: "+ aireAttendue);
    message.channel.send("centre de gravité attendu: "+ centreAttendu);
    message.channel.send("présence de vache attendue: "+
presenceAttendue);
    message.channel.send("*** Remplissage terminé ! ***");
    message.channel.send("`!lancement` pour lancer les calculs");
    message.channel.send("`!etat` pour visualiser l'enclos");
    }
    },
    name: "essai",
};
};

```

etat.js

```
/**
 * @file etat.js
 * @usage `!etat`
 * @fileoverview Commande d'affichage du contenu actuel de l'enclos
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/** requisition des modules nécessaires */
let noiraude = require('../libs/noiraude');
let io = require('../libs/io')

/** Exportation du module local en commande appelée via son attribut `name` */
module.exports = {
  execute(message, args){
    message.channel.send(
      "````" + io.jsonPrettifier(noiraude.enclos) + "````"
    );
  },
  name: "etat",
};
```



hello.js

```
/**
 * @file hello.js
 * @usage `!hello`
 * @fileoverview Affichage d'un message de salutation aléatoire
 * @author Julien Mocquet
 */

/**
 * Tableau de chaîne de caractères de messages d'accueil
 * @type {Array}
 * @const
 */
const listHello = [
  'Hello les Loulous',
  'Bonjour les amis',
  "Allo Docteur ici la Noiraude",
];

/** Exportation du module local en commande appelée via son attribut `name` */
module.exports = {
  execute(message, args){
    /**
     * Tirage d'un message aléatoire
     * `Math.floor` - suppression décimale
     * `Math.random` Tirage aléatoire dans l'intervalle ]0;1[
     */
    message.channel.send(
      `${listHello[Math.floor(Math.random() * listHello.length)]}`);
  },
  name: "hello",
};
```

```
/**
 * @file info.js
 * @usage `!info`
 * @fileoverview Affiche les informations sur le projet
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/**
 * Chaîne de caractères brute des informations sur le projet
 * @type {String}
 * @const
 */
const info = `
    **La noiraude** est un robot discord de calcul de centre de gravité réalisé
    par:
    - Théo Gicquel https://gitlab.com/TheoGicquel
    - Julien Mocquet https://gitlab.com/jujukibus

    Réalisé du 02 Avril 2021 au 17 Mai 2021
    Le code source de ce projet est consultable sur
https://gitlab.com/TheoGicquel/la-noiraude-bot
    version : ** release v2.0 - Vieux Lille :cheese: **
`;

/** Exportation du module local en commande appelée via son attribut `name` */
module.exports = {
    execute(message, args){
        message.channel.send(info);
    },
    name: "info",
};
```

```

/**
 * @file lancement.js
 * @usage `!lancement`
 * @fileoverview lance les différents calculs afin d'afficher le centre de
 * gravité de l'enclos
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/** requisition des modules nécessaires */
let mathAPI = require('../libs/mathAPI.js');
let noiraude = require('../libs/noiraude');
let io = require('../libs/io');

/** Exportation du module local en commande appelée via son attribut `name` */
module.exports = {
  execute(message, args){
    /** Contrôle du nombre d'arguments */
    if(noiraude.enclos.length<3){
      return message.reply("Définissez l'enclos d'abord !");
    }

    if(noiraude.enclos.length>noiraude.maxPiquets){
      return message.reply(
        "L'enclos saisi possède trop de piquets ! ( max : " +
        noiraude.maxPiquets + ")"
      );
    }

    /** Message invitant l'utilisateur à attendre les résultats */
    message.channel.send("Calculs en cours... veuillez patienter :watch:");

    /** calcul de l'aire à partir de l'enclos */
    let aire = mathAPI.getAirePolygone(noiraude.enclos);

    /** contrôle de validation du calcul de l'aire */
    if(isNaN(aire))
    {
      console.error("Aire NaN");
      message.channel.send(":x:**Erreur calcul **: aire ");
    }

    /** calcul du centre de gravité à partir de l'aire et de l'enclos */

    let centreGravite = mathAPI.getCentreGravite(noiraude.enclos,aire);
    /** contrôle de validation des calculs du centre de gravité */
    if(isNaN(centreGravite.x) || isNaN(centreGravite.y))
    {
      console.error("centregrav NaN");
      message.channel.send(":x:**Erreur calcul **: centre de gravité ");
    }

    /** calcul de l'appartenance du gravité à l'enclos */
    let appartenance = mathAPI.getAppartenancePointPolygone(
      noiraude.enclos,centreGravite
    );
  }
};

```

```

/** Affichage des résultats des calculs */
message.channel.send("aire obtenue : " + mathAPI.getValAbsolue(aire));
message.channel.send(
    "***centre de gravité** : " +
    "```" + io.jsonPrettifier(centreGravite) + "```"
);

switch (appartenance) {
    case true:
        message.channel.send(
            "*** La vache est dans le pré !** :herb: :cow: :herb:"
        );
        break;
    case false:
        message.channel.send(
            "*** La vache est en dehors du pré !** :cow: :herb: :herb:"
        );
        break;

    default:
        message.channel.send(
            "*** La vache est perdue.. ** :cow: :cloud_tornado:"
        );
        console.error("appartenance invalide")
        break;
}

return message.channel.send("calculs terminés ! :checkered_flag: ");
},
name: "lancement",
};

```

raz.js

```
/**
 * @file raz.js
 * @exports `!raz`
 * @fileoverview Commande de remise à zéro de l'enclos
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/** requisition du module de l'enclos */
let noiraude = require('../libs/noiraude');

/** Exportation du module local en commande appelée via son attribut `name` */
module.exports = {
  execute(message, args){
    noiraude.enclos={};
    return message.channel.send("Enclos effacé !");
  },
  name: "raz",
};
```

saisie.js

```
/**
 * @file saisie.js
 * @usage `!saisie`
 * @fileoverview Commande de saisie de multiples piquets via une seule commande
 * @author Theo Gicquel <theo.gicquel.work@gmail.com>
 */

/** requisition des modules nécessaires */
let noiraude = require('../libs/noiraude');
let io = require('../libs/io');
let saisieHelpText="Format :\n `!saisie x,y:x,y etc..."
let validation;

/** Exportation du module local en commande appelée via son attribut `name`*/
module.exports = {
  execute(message,args){
    switch (args.length) {
      case 1:
        validation = io.saisievalidation(args[0]);
        switch (validation) {
          case true:
            noiraude.enclos=io.saisieParser(args[0]);
            return message.channel.send(
              ":white_check_mark: Saisie effectuée !\n"
              + "````" + io.jsonPrettifier(noiraude.enclos)
              + "````"
            );
          case false:
            return message.channel.send(":x: Saisie invalide !");
          default:
            return message.channel.send(
              ":thinking: Problème de saisie innatendu"
            );
        }
      default:
        message.channel.send(
          `Nombre d'arguments incorrects ! ${args.length} au lieu de
2`);
        return message.channel.send(saisieHelpText);
    }
  },
  name: "saisie",
};
```