Thauvin OSEL
Sion HAN
Maël HUBERT
Victoire BESCHU

La machine de dames

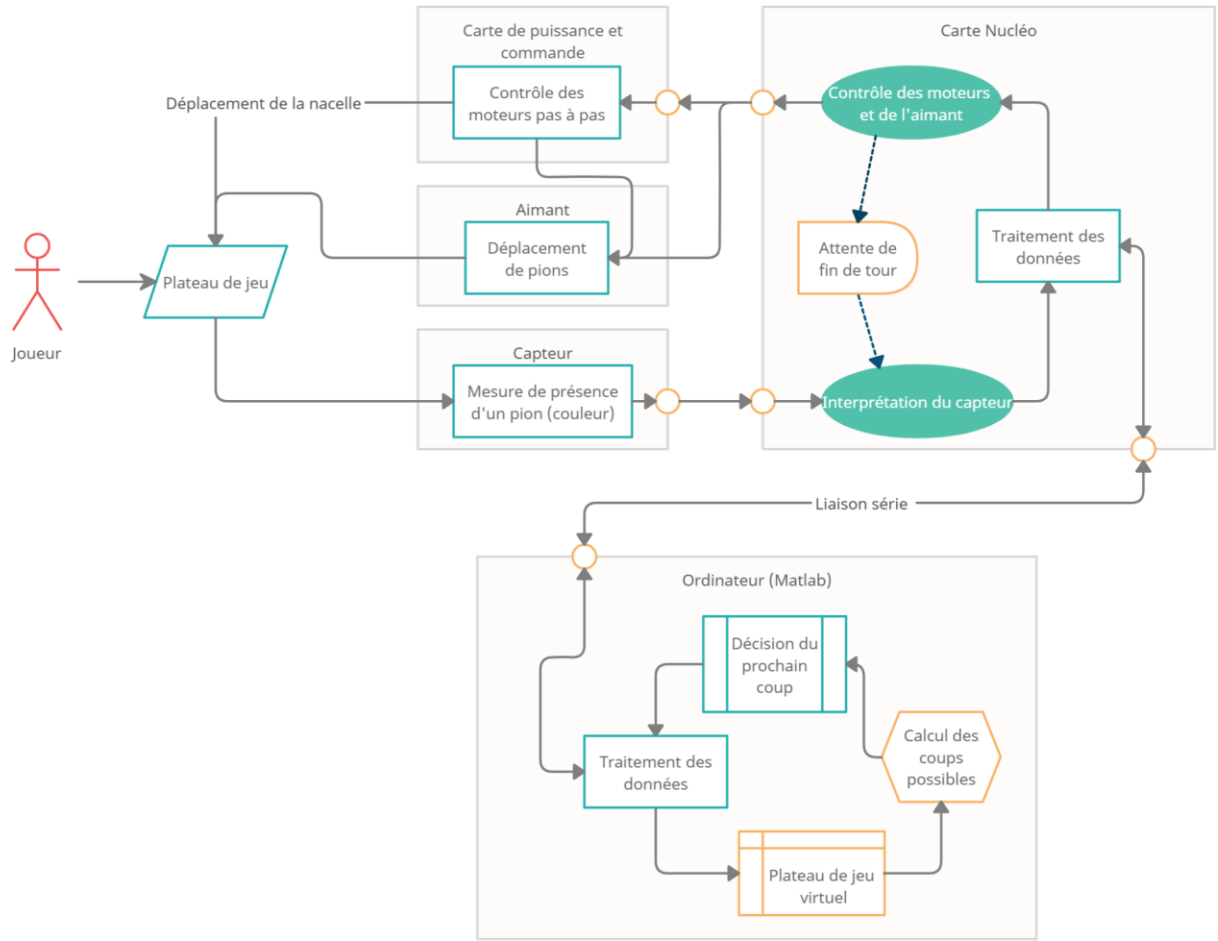
Objectifs :

- Créer un jeu de dames totalement mécanisé

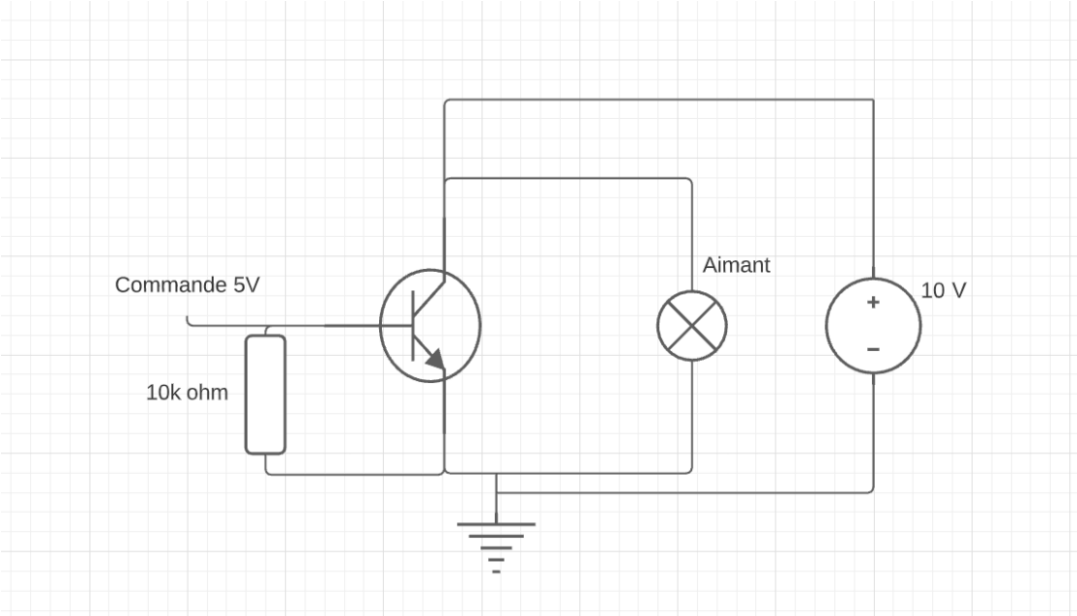
Cahier des charges :

- ✔ Pouvoir déplacer un pion d'une case à une autre case quelconque, et ceci dans un délai raisonnable pour une partie réelle
- ✔ Quadriller la surface fonctionnelle du bras en un damier
- ✔ Calculer quel coup a faire pour l'ordinateur
- ✔ Distinguer les pions des deux équipes
- ✔ Détecter quel coup est fait par l'humain

Schéma fonctionnel :



I/ Déplacement des pions à l'aide d'un électroaimant



Branchage des cartes de puissance et de commande:

- Vmot: directement sur une source de tension constante
- Gnd: Directement au sol
- Clock: Sur la carte. Un front montant est équivalent à un pas du moteur 5V: sur le 5V de la carte
- Enable: Sur la carte. Si il n'y a pas de tension dans cette entrée, le moteur ne fonctionne pas.
- CW: Sur la carte. Si il n'y a pas de tension, le moteur va dans un sens. Si il y a une tension, il va dans l'autre sens.

Les autres entrées ne sont pas nécessaires.

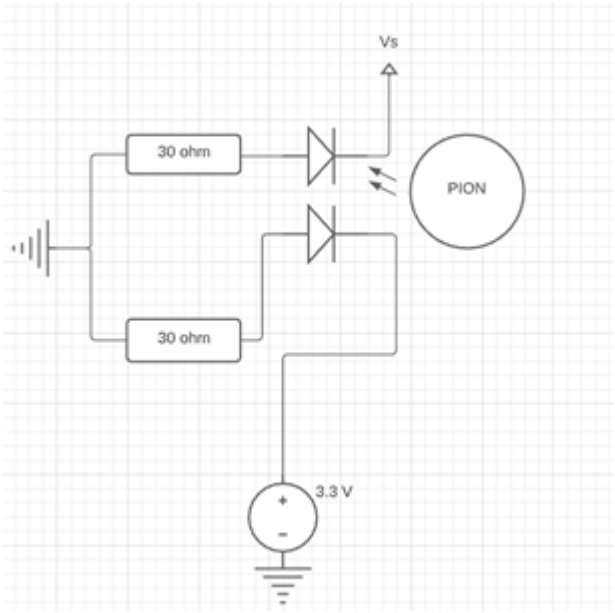
Pour assurer un déplacement des pions facile, nous avons choisi d'utiliser des pions aimantables afin de pouvoir simplement piloter un électroaimant rattaché au bras de la table traçante.

Quelques informations sur l'électroaimant :

- La dimension : l'électroaimant ne doit pas être plus grand qu'une case afin de ne pas potentiellement empiéter sur une autre case et donc accidentellement déplacer d'autres pions.
- La chauffe : l'un des défauts de l'électroaimant est la chauffe assurée, il est donc impossible de le toucher à main nue car même si il alterne entre être alimenté lorsqu'il aimante un pion et non alimenté lorsqu'il doit déposer le pion sur une case, il chauffe facilement.

II/ Photodétection

Il nous faut un capteur capable de différencier les pions des 2 équipes et les différents états de cases (occupé ou non). Pour ce faire, nous avions le choix entre une caméra ou un capteur de couleur.



Pour réaliser ce capteur, on utilise un montage vu en TP d'électronique lors du semestre 5 à l'aide d'une diode et d'une photodiode. La lumière de la photodiode se réfléchit du pion vert et est captée par la diode. On mesure à l'oscilloscope Vs, qui dépend directement de ce que la diode capte. On établit des valeurs seuils pour Vs qui correspondent à différentes couleurs :

Case blanche	Pion vert	Case noire
Vs > 700 mV	300 mV <Vs< 500 mV	Vs < 200 mV

III/ Programme côté Matlab

Le programme Matlab, connecté par liaison série à la carte Nucléo, sert à décider du prochain coup pris par l'ordinateur. Cette liaison série est utilisée par l'intermédiaire de *getMove*, qui envoie une liste de coups valides possibles jouées par l'humain, et reçoit le coup joué par l'humain, et de *writeMove*, qui envoie à la carte le coup à jouer. Internement, la fonction *nextMoves* permet de donner les

coups possibles d'un joueur donné selon un plateau donné, dont la sortie est donnée à la fonction *minimax* qui utilise un algorithme légèrement optimisé de minimisation des pertes maximales, donnant en sortie le coup « optimal » pour une profondeur de calcul donnée. De plus, on a la fonction *showBoard* qui permet de directement visualiser le plateau de jeu sur ordinateur, si besoin. Les autres fonctions sont utilisées par ci par là dans le code en tant qu'utilitaires.

III/ Programme côté carte Nucléo

Le programme C de la carte s'occupe principalement du déplacement de la nacelle, du contrôle de l'électroaimant et de la détection. Ce programme communique avec le programme Matlab à plusieurs moments : pour l'initialisation, carte récupère le joueur joué par l'ordinateur, donné par la fonction principale de Matlab, et dans la suite du programme, la fonction *cpuTurn* communique avec Matlab pour effectuer le tour de l'ordinateur, et la fonction *getMove* récupère la liste de coups possibles joués par l'humain donné par Matlab, et renvoie la position jouée. Ces fonctions se basent sur les fonctions auxiliaires de calcul, mais surtout sur les fonctions de déplacement de la nacelle et de contrôle de l'électroaimant, tel que *move*, et de détection en utilisant le capteur, tel que *detection*.

Planning + Carnet de bord

02 03 2021

- Bouton de fin de tour
- Déplacer le bras à une position x,y donnée (calibration)

L'équipe a globalement fini d'étudier toute documentation technique, et Maël a bien entamé la partie programmation.

07 03 2021

- Détecter l'état de la case (rouge/noir/vide/dame…….)

Maël et Osel ont travaillé sur le programme : débuggage de celui-ci, test de quelques commandes simples. Osel a aussi fait fonctionner les moteurs pas à pas.

Sion et Victoire ont repris le TP du semestre 5 sur le montage de photodétection : réalisation du montage puis test de celui-ci.

14 04 2021

- Déplacer les pions ? (pions aimantés + aimant commandé par la carte) (partie physique)// à discuter avec prof pour le matériel
- Enlever les pions mangés (faire un programme qui amène un pion à un endroit spécifique en dehors du plateau)

Continuité de la séance précédente. Des problèmes sur la liaison carte nucléo - matlab, qui a considérablement freiné notre avancée.

05 05 2021

- Déplacer les pions (pions métalliques + aimant commandé par la carte) (partie électrique + programmation)

Osel a trouvé un plateau de jeu et des pions aimantables. Le montage complet a donc pu être testé, avec toujours Sion et Victoire sur la partie électronique (ici l'électroaimant) et Osel et Maël sur la partie informatique. Les programmes de commandes simples marchent bien, comme transporter un pion d'une case à une autre.

18 05 2021

- tout finir (troubleshooting et optimisation possible)+ préparation exposé
- Dessin vectoriel ? (en cas de pépin)

Tentative à 4 de faire fonctionner le programme complet de Maël, qui modélise le plateau entier et permet de jouer contre un IA qui fait des mouvements aléatoires. Malheureusement il ne marchera pas : le bras ne s'initialise pas correctement à la case 0, et le bras ne se déplaçait plus bien en diagonal à partir d'un moment. La cause était un défaut dans la liaison physique au niveau d'un moteur.

Annexe 1 : Programme Matlab

Thauvin OSEL
Sion HAN
Maël HUBERT
Victoire BESCHU

Main_Card :

```
% nucleo = serialport('COM9', 9600);     %(A faire manuellement pour éviter les bugs)

player = input('Quel joueur sera le CPU? Bleu = 0, Rouge = 1\n');
depth = input('Quelle profondeur de calcul pour le CPU?\n');

writeline(nucleo,int2str(player));

gameState = 0;
Board = initBoard();
turn = 1;
score = 0;


showBoard(Board,score);
nMc = nextMoves(Board,player);
nMh = nextMoves(Board,1-player)

while (gameState == 0)
   if (mod(turn-1,2)==player)
      move = minimax(Board,player,nMc,depth)  % nMc(randi(length(nMc)))
      writeMove(move,nucleo);
   else
      check = 0;
      while (check==0)
         pause;
         try
            move = getMove(nMh,nucleo); % input('Entrez votre coup\n','s')
            check = 1;
         catch
            disp('Coup invalide, tapez votre coup ou appuyez sur entrée pour réessayer');
            move = input('Entrez votre coup\n','s');
            if (~strcmp(move,""))&&(any(ismember(move,nMh)))
               check = 1;
               move
            end
         end
      end
   end

   Board = applyMove(Board,move);
   showBoard(Board,score);

   nMc = nextMoves(Board,player);
   nMh = nextMoves(Board,1-player)
```

```
    score = evalScore(Board,1-player,nMh,nMc);
    showBoard(Board,score);

%     Board

    if (isempty(nMc))
        gameState = -1;
    elseif (isempty(nMh))
        gameState = 1;
    end

    turn = turn + 1;
end

res = (1+gameState)/2;
fprintf("Le joueur %d a gagné!\n",res);

%clear nucleo;
```

posLCtoM:

```
function m = posLCtoM(x,y)
%Syntax
%  posLCtoM(x,y)
%Description
%  Traduit une position ligne/colonne en une notation Manoury sur plateau
%  Marche sur des entiers ou des tableaux d'entiers

m = 5*(x-1) + (y+(mod(x,2)==0))/2;

end
```

posMtoLC:

```
function [x,y] = posMtoLC(m)
%Syntax
%  posMtoLC(m)
%Description
%  Traduit une position d'une notation Manoury en ligne/colonne sur plateau
%  Marche sur un entier ou un tableau d'entier

if any((m<0)|(m>50))
```

```
    error("Position invalide");
else
    x = 1 + fix((m-1)/5);
    y = 2*(rem(m,5)) + (10-(rem(m,10)==0)).*(rem(m,5)==0) + (rem(m,10)<=5) - 1;
    % y = 2*rem(m,5) - rem(fix(m/5),2);
end
end
```

initBoard:

```
function [Board] = initBoard()
%Syntax
%  showBoard(Board)
%Description
%  Initialise le plateau

Board = zeros(10);
[x1,y1] = posMtoLC(31:50);
[x2,y2] = posMtoLC(1:20);
for i = 1:20
    Board(x1(i),y1(i)) = 1;
    Board(x2(i),y2(i)) = -1;
end
end
```

showBoard:

```
function showBoard(Board,score)
%Syntax
%  showBoard(Board)
%Description
%  Affiche et met à jour le plateau

r = 5;

M = 2*r*(~mod((1:10)+(1:10)',2))-r;
[x1,y1] = find(Board == 1); % Indices de pions du joueur 1
[x1d,y1d] = find(Board == 2); % Indices des dames du joueur 1
[x2,y2] = find(Board == -1); % Indices de pions du joueur 2
[x2d,y2d] = find(Board == -2); % Indices des dames du joueur 2

figure(1)
hold on
imagesc(M)
```

Thauvin OSEL
Sion HAN
Maël HUBERT
Victoire BESCHU

```
n = 10*100;
C = [160/255 82/255 45/255];
for i = 2:(n-1)
    if (-score>2*r*(i/n - 0.5))
        C = [C;[1 0 0]];
    else
        C = [C;[0 0 1]];
    end
end
C = [C;[1 1 1]];

colormap(flipud(C))
h = colorbar;
title(h,"Ton score: " + int2str(score))

[x,y] = posMtoLC(1:50);
for i = 1:50
    text(y(i)-0.5,11-x(i)-0.4,int2str(i))
end

for i = 1:length(x1)
    p = nsidedpoly(100, 'Center', [y1(i) 11-x1(i)], 'Radius', 0.4);
    plot(p, 'FaceColor', 'b', 'FaceAlpha', 1)
end
for i = 1:length(x1d)
    p = nsidedpoly(100, 'Center', [y1d(i) 11-x1d(i)], 'Radius', 0.4);
    plot(p, 'FaceColor', 'c', 'FaceAlpha', 1)
end
for i = 1:length(x2)
    p = nsidedpoly(100, 'Center', [y2(i) 11-x2(i)], 'Radius', 0.4);
    plot(p, 'FaceColor', 'r', 'FaceAlpha', 1)
end
for i = 1:length(x2d)
    p = nsidedpoly(100, 'Center', [y2d(i) 11-x2d(i)], 'Radius', 0.4);
    plot(p, 'FaceColor', 'm', 'FaceAlpha', 1)
end

ylim([0.5 10.5])
xlim([0.5 10.5])
axis square
set(gca,'visible','off')
set(gca,'xtick',[])
hold off

end
```

Thauvin OSEL
Sion HAN
Maël HUBERT
Victoire BESCHU

nextMoves:

```
function possMoves = nextMoves(Board,player)
%Syntax
%  possMoves = nextMoves(Board,player)
%Description
%  Donne les coups légaux d'un joueur

r = (-1)^(player); % Utile

[x0,y0] = find(Board == r); % Indices de pions du joueur
[x0d,y0d] = find(Board == 2*r); % Indices des dames du joueur

possT = [];
possM = [];

inBounds = @(i,j) (i>=1)&&(j>=1)&&(i<=10)&&(j<=10);

   function res = pieceTake(i,j) % Parcours en profondeur récursif
      res = [];

      if (inBounds(i-2*r,j-2*r)&&(sign(Board(i-r,j-r))==-r)&&(Board(i-2*r,j-2*r)==0)) % Prise sur
diagonale \
         t = pieceTake(i-2*r,j-2*r);
         if isempty(t)
            res = [res (string(posLCtoM(i,j)) + "x" + string(posLCtoM(i-2*r,j-2*r)))];
         else
            res = [res (string(posLCtoM(i,j)) + "x" + t)];
         end
      end

      if (inBounds(i-2*r,j+2*r)&&(sign(Board(i-r,j+r))==-r)&&(Board(i-2*r,j+2*r)==0)) % Prise
sur diagonale /
         t = pieceTake(i-2*r,j+2*r);
         if isempty(t)
            res = [res (string(posLCtoM(i,j)) + "x" + string(posLCtoM(i-2*r,j+2*r)))];
         else
            res = [res (string(posLCtoM(i,j)) + "x" + t)];
         end
      end

   end

for i = 1:length(x0)
```

```
    x = x0(i);
    y = y0(i);

    take = pieceTake(x,y);
    if (~isempty(take))
        possT = [possT take];

    else
        if inBounds(x-r,y-r)&&(Board(x-r,y-r)==0) % Diagonale \ libre
            possM = [possM (string(posLCtoM(x,y)) + "-" + string(posLCtoM(x-r,y-r)))];
        end
        if inBounds(x-r,y+r)&&(Board(x-r,y+r)==0) % Diagonale / libre
            possM = [possM (string(posLCtoM(x,y)) + "-" + string(posLCtoM(x-r,y+r)))];
        end
    end
end


    function [k,t] = diagAux(i,j,ri,rj)
        k = 0;
        t = 0;
        while inBounds(i+ri*k,j+rj*k)&&(t==0)
            if inBounds(i+ri*(1+k),j+rj*(1+k))
                if (sign(Board(i+ri*(1+k),j+rj*(1+k)))==-r) % Pièce ennemie dans le passage
                    if
inBounds(i+ri*(2+k),j+rj*(2+k))&&(Board(i+ri*(2+k),j+rj*(2+k))==0)&&(~vu(posLCtoM(i+ri*(1+k
),j+rj*(1+k)))) % Pièce prenable
                        t = 2;
                    else % Pièce non prenable
                        t = 1;
                    end
                elseif (sign(Board(i+ri*(1+k),j+rj*(1+k)))==r) %  Pièce alliée dans le passage
                    t = 1;
                end
            end
            k = k+1;
        end
        %switch t
        %    case 2
        %        k = k+1;
        %    otherwise
        %        k = k-1;
        %end
        k = k+1;
    end
```

```
    function res = diagKingTake(i,j,ri,rj)
        res = [];
        [k,t] = diagAux(i,j,ri,rj); % Détermine la position et le type d'arrêt selon la diagonale

        if (t==2) % Pièce prenable dans le passage
            vu(posLCtoM(i+ri*(k-1),j+rj*(k-1))) = 1;
            res = [res (string(posLCtoM(i,j)) + "x" + kingTake(i+ri*k,j+rj*k))]; % Récursion car
prise obligatoire
        else % Arrêt forcé (pièce imprenable ou mur dans le passage)
            %for l = 1:(k-2) % Choix de la case d''arrêt
            res = [string(posLCtoM(i,j))]; %+ "-" + string(posLCtoM(i+ri*l,j+rj*l))
            %end
        end
    end


    function res = kingTake(i,j) % Parcours en profondeur récursif
        res = [];
        %fprintf("%d,%d\n",i,j); C'était lui ptn
        res = [res diagKingTake(i,j,1,1)]; % Prise sur diagonale <\
        res = [res diagKingTake(i,j,-1,-1)]; % Prise sur diagonale \>
        res = [res diagKingTake(i,j,-1,1)]; % Prise sur diagonale </
        res = [res diagKingTake(i,j,1,-1)]; % Prise sur diagonale />
    end

    function kingMove(i,j,ri,rj)
        k = 1;
        res = "";
        while (inBounds(i+ri*k,j+rj*k)&&(Board(i+ri*k,j+rj*k)==0)) % Diagonale libre
            possM = [possM (string(posLCtoM(i,j)) + "-" + string(posLCtoM(i+ri*k,j+rj*k)))];
            k = k+1;
        end
    end

for i = 1:length(x0d)  % Dames
    x = x0d(i);
    y = y0d(i);

    vu = zeros(50,1);

    take = unique(kingTake(x,y)); % Epuration de la liste

    temp = []; % Retrait des cas d'arrêt seuls, sans vrai coup
    for i = 1:length(take)
        if (~isnan(str2double(take(i))))
            temp = [temp i];
```

```
        end
    end
    take(temp) = [];

    if (~isempty(take))

        temp = []; % Prise la plus longue obligatoire
        for i = take
            temp = [temp length(split(i,"x"))];
        end
        take = take(temp==max(temp));
        assignin('base','take',take)

        for i = 1:length(take) % Arrêt au choix après une prise
            [x,y] = posMtoLC(str2double(split(take(i),"x")'));
            n = length(x);
            ri = sign(x(n)-x(n-1));
            rj = sign(y(n)-y(n-1));

            possT = [possT (take(i))];
            k = 1;
            res = "";
            while (inBounds(x(n)+ri*k,y(n)+rj*k)&&(Board(x(n)+ri*k,y(n)+rj*k)==0)) % Diagonale
libre
                possT = [possT (take(i) + "-" + string(posLCtoM(x(n)+ri*k,y(n)+rj*k)))];
                k = k+1;
            end
        end
    else
        kingMove(x,y,1,1); % Mouvement sur diagonale <\
        kingMove(x,y,-1,-1); % Mouvement sur diagonale \>
        kingMove(x,y,-1,1); % Mouvement sur diagonale </
        kingMove(x,y,1,-1); % Mouvement sur diagonale />
    end
end

assignin('base','possM',possM) % Debug
assignin('base','possT',possT)

if isempty(possT)
    possMoves = possM;
else % Prise obligatoire
    temp = [];
    for i = possT
        temp = [temp length(split(i,"x"))];
    end
```

```
    possMoves = possT(temp==max(temp)); % Rajouter le find si erreur
end
end
```

evalScore:

```
function score = evalScore(Board,player,nMp1,nMp2)
%Syntax
%  score = evalScore(Board,player,nMp1,nMp2)
%Description
%  Donne le score d'un joueur

r = (-1)^(player);

if (isempty(nMp1))
    score = -inf;
elseif (isempty(nMp2))
    score = +inf;
else
    sp = nnz(Board==r)-nnz(Board==-r);
    sd = 2*(nnz(Board==2*r)-nnz(Board==-2*r));
    score = sp + sd;
end
end
```

minimax:

```
function bestMove = minimax(Board,player,possMoves,depth)
%Syntax
%  bestMove = minimax(Board,player,possMoves,depth)
%Description
%  Donne le meilleur coup au sens le l'algorithme minimax alpha-beta pruning adapté en
negamax

    function score = negamax(Board,depth,player,alpha,beta)
        nMp1 = nextMoves(Board,player);
        nMp2 = nextMoves(Board,1-player);

        nd = nnz(abs(Board)==2);
        depth = fix(depth*(1-(nd/4<depth)*nd/4)); % A revoir

        if (depth==0)||(isempty(nMp1))||(isempty(nMp2))
            score = evalScore(Board,player,nMp1,nMp2); % Return
```

```
        else
            score = -inf;
            for m = nMp1
                newBoard = applyMove(Board,m);
                auxScore = -negamax(newBoard,depth-1,1-player,-beta,-alpha);
                if (auxScore>=score)
                    score = auxScore;
                end
                if (auxScore>alpha)
                    alpha = auxScore;
                end
                if (alpha>=beta)
                    score = alpha; % Return alpha
                end
            end
            % Return score
        end
    end

n = length(possMoves);
if (n==1)
    bestMove = possMoves(1);
else
    tabScores = zeros(1,n);
    alpha = -inf;
    for i = 1:n
        tabScores(i) = -negamax(applyMove(Board,possMoves(i)),depth-1,1-player,alpha,-
alpha);
    end
    bestIndexes = find(tabScores == max(tabScores));
    bestMove = possMoves(bestIndexes(randi(length(bestIndexes))));
end
end
```

getMove:

```
function move = getMove(possMoves,nucleo)
%Syntax
%  [x,y] = getMove(possMoves,inputArg2)
%Description
%  Demande, lis puis renvoie la dernière position jouée sur le plateau

n = length(possMoves);

check = 0;
```

```
while (check ~= 1221) % Attente de connexion
    check = str2double(readline(nucleo));
end

writeline(nucleo,int2str(n)); % Transmission de la longueur des données

for p = possMoves % Transmission des positions possibles
    writeline(nucleo,length(p)); % Debug string
    writeline(nucleo,p);
end

i = 0;
while (i==0) % Récéption de la position
    i = str2double(readline(nucleo));
end

if (i~=n+1) % Retourne la position jouée
    move = possMoves(i);
else
    error('Position jouée invalide');
end

end
```

writeMove:

```
function writeMove(m,nucleo)
%Syntax
%  writeMove(inputArg1,inputArg2)
%Description
%  Ecrit une position sur le plateau

check = "";
while (check == "") % Attente de connexion
    check = readline(nucleo);
end

writeline(nucleo,length(m)); % Debug string
writeline(nucleo,m); % Transmission du coup

end
```

Thauvin OSEL
Sion HAN
Maël HUBERT
Victoire BESCHU

Annexe 2 : Programme Nucléo

```
/* Déclaration des ressources externes */
#include "mbed.h"

/* Déclaration des variables globales */
#define T 0.01
#define xr 333 /* WIP */
#define yr 333 /* WIP */


/* Déclaration des entrées/sorties */
PwmOut Clock1(D7);
DigitalOut CW1(D6);
DigitalOut Enable1(D5);
PwmOut Clock2(D4);
DigitalOut CW2(D3);
DigitalOut Enable2(D2);
DigitalOut Magnet(D8);
AnalogIn Capteur(A0);
Serial matlab(USBTX, USBRX);

/* Déclaration de fonctions */

void initClock();

void initPos();

int initPlayer();

int fmax(int nb1, int nb2);

int fmin(int nb1, int nb2);
```

```
void tourne1(int s, int nb);

void tourne2(int s, int nb);

void tourne(int s1, int nb1, int s2, int nb2);

void posMtoLC(int p, int* x, int* y);

int posLCtoM(int x, int y);

void move(int p0, int p, bool a);

void cpuTurn(); /* à test */

int detection(); /* WIP */

void getMove(int player); /* à test */


/* Fonction principale */

int main(){
   int player, turn;

   initClock();
   initPos();
   matlab.baud(9600);

   player = initPlayer();
   turn = 1;

   while (1) {
      if (((turn-1)%2) == player) {
         cpuTurn();
      }
      else {
         getMove(player);
      }
      turn = turn + 1;
   }
}


/* Fonctions secondaires */

void initClock() { /* Initialise les clock moteurs */
```

```
    Clock1.period(T);
    Clock1.write(1);
    Clock2.period(T);
    Clock2.write(1);
}

void initPos() { /* Initialise la position */
    move(1,51,0); /* WIP */
}

int initPlayer() {  /* Récupère et initialise l'indice de joueur CPU */
    int f = -1;
    while (f == -1) {
        if (matlab.readable()) {
            matlab.scanf("%d",&f);
        }
    }
    return f;
}

int fmax(int nb1, int nb2){ /* Calcul auxiliaire */
    int res;
    if (nb1>=nb2) {res = nb1;}
    else {res = nb2;}
    return res;
}

int fmin(int nb1, int nb2){ /* Calcul auxiliaire */
    int res;
    if (nb1<=nb2) {res = nb1;}
    else {res = nb2;}
    return res;
}

void tourne1(int s, int nb){ /* Mouvement selon moteur 1 */
    Enable1 = 1;
    CW1 = s;

    Clock1.write(0.8);
    wait(nb*T);
    Clock1.write(1);

    Enable1 = 0;
}
```

```
void tourne2(int s, int nb){ /* Mouvement selon moteur 2 */
    Enable2 = 1;
    CW2 = s;

    Clock2.write(0.8);
    wait(nb*T);
    Clock2.write(1);

    Enable2 = 0;
}


void tourne(int s1, int nb1, int s2, int nb2){ /* Mouvement quelconque */
    double Tm;

    if (nb2 == 0) { /* Mouvement selon moteur 1 */
        tourne1(1-s1,nb1);
    }
    else if (nb1 == 0) { /* Mouvement selon moteur 2 */
        tourne2(1-s2,nb2);
    }
    else { /* Mouvement diagonal */
        CW1 = s1;
        CW2 = s2;

        Enable1 = 1;
        Enable2 = 1;

        Tm = T*(fmax(nb1,nb2)*1.0 / fmin(nb1,nb2));

        if (fmax(nb1,nb2) == nb1){
            Clock1.period(Tm);
            Clock1.write(0.8);
            Clock2.write(0.8);
            wait(nb1*T);
            Clock1.write(1);
            Clock2.write(1);
            Clock1.period(T);
        }
        else {
            Clock2.period(Tm);
            Clock1.write(0.8);
            Clock2.write(0.8);
            wait(nb2*T);
            Clock1.write(1);
            Clock2.write(1);
```

```
            Clock2.period(T);
        }

        Enable1 = 0;
        Enable2 = 0;
    }
}

void posMtoLC(int p, int* x, int* y) { /* Traduit une position p en (x,y) */
    *x = 1 + floor((p-1)/5.0); /* trunc -> floor */
    *y = 2*(p%5) + (10-((p%10)==0))*((p%5)==0) + ((p%10)<=5) - 1;
    /* int temp = floor(p/5.0); *y = 2*(p%5) - (temp%2); */
}

int posLCtoM(int x, int y) { /* Traduit une position (x,y) en p */
    int p = 5*(x-1) + (y+(x%2==0))/2;
    return p;
}


void move(int p0, int p, bool a) {
    int x,y,x0,y0,s1,s2;
    posMtoLC(p,&x,&y);
    posMtoLC(p0,&x0,&y0);

    s1 = ((x-x0) > 0) - ((x-x0) < 0);
    s2 = ((y-y0) > 0) - ((y-y0) < 0);

    if (a) {
        Magnet = 1;
    }
    tourne(1-s1,abs(x-x0)*xr,1-s2,abs(y-y0)*yr);
    Magnet = 0;
}

void cpuTurn() {
    int i,n;
    int p,p0,p1;
    int x0,y0,x1,y1,s1,s2;
    int n0;
    char* s = "";

    if (matlab.writeable()) { /* Initialisation de la connexion, Matlab attend carte */
        matlab.printf("1001\n");
    }
```

```
while (s == "") { /* Réception des données (Utiliser s.compare("")?) */
    if (matlab.readable()) {
        matlab.scanf("%d",&n0); /* Taille du string */
        matlab.gets(s,n0); /* scanf : bizzare avec le char [] ou char* */
    }
    wait(0.1);
}

p0 = 10*s[0]+s[1];
move(51,p0,0); /* Déplacement vers le pion */

if (s[2] == '-') { /* Déplacement */
    p = 10*s[3]+s[4];
    move(p0,p,1);
}
else { /* Prise */
    n = (strlen(s)+1)/3;
    for (i=1;i<n;i++) { /* Déplacement du pion prenant */
        p = 10*s[3*i]+s[3*i+1];
        move(p0,p,1);
        p0 = p;
    }


    for (i=1;i<n;i++) { /* Déplacement des pions pris hors du plateau */
        p1 = 10*s[3*i]+s[3*i+1];

        posMtoLC(p0,&x0,&y0);
        posMtoLC(p1,&x1,&y1);

        s1 = ((x0-x1) > 0) - ((x0-x1) < 0);
        s2 = ((y0-y1) > 0) - ((y0-y1) < 0);

        p = posLCtoM(x1-s1,y1-s2); /* Selection du pion pris */

        if (i==1) { /* Initialisation */
            move(p0,p,0); /* Déplacement vers pion */
        }
        else { /* Cas général */
            move(51,p,0); /* Déplacement vers pion */
        }

        move(p,51,1); /* Déplacement en dehors du plateau */

        p0 = p1;
    }
```

```
    }

    move(p0,51,0); /* Remise à zéro */
}

int detection() {
    int i;
    double val = 0.0;
    int res = 0;
    double moy = 0.0;

    for (i=0;i<100;i++) {
        val = Capteur.read();
        moy = moy + val/100;
        wait(0.25/100);
    }
    moy = moy/1; /* A changer selon fonctionnement */

    if (moy<0.23) { /* Joueur 2 */
        res = -1;
    }
    else if ((moy>0.23)&&(moy<0.4)) { /* Joueur 1 */
        res = 1;
    }
    /* Sinon case vide */

    return res;
}

void getMove(int player) {
    int i,j,n,ns;
    int p,p0,p1;
    int x0,y0,x1,y1,s1,s2;
    int n0;
    char* s;
    char** tab;
    bool res, res2;

    if (matlab.writeable()) { /* Initialisation de la connexion, Matlab attend carte */
        matlab.printf("1221");
    }

    if (matlab.readable()) {
        matlab.scanf("%d",&n);
    }
```

```
   tab = (char**) malloc(3*n*sizeof(char)); /* A voir si c'est la syntaxe correcte */
   for (i=0;i<n;i++) { /* Utiliser s.compare("")? */
      if (matlab.readable()) {
         matlab.scanf("%d",&n0);
         matlab.gets(tab[n],n0);
      }
   }

   s = tab[0];
   p0 = 10*s[0]+s[1];
   move(51,p0,0); /* Déplacement vers le pion */

   i = 0;
   res = 0;
   while (not(res)&&(i<n)) {
      s = tab[i];

      if (s[2] == '-') { /* Déplacement */
         p = 10*s[3]+s[4];
         move(p0,p,0);
         res = (detection()==player);
      }
      else { /* Prise */
         ns = (strlen(s)+1)/3;
         j = 1;
         res2 = 1;

         while (res2&&(j<ns)) {
            p1 = 10*s[3*j]+s[3*j+1];

            posMtoLC(p0,&x0,&y0);
            posMtoLC(p1,&x1,&y1);

            s1 = ((x0-x1) > 0) - ((x0-x1) < 0);
            s2 = ((y0-y1) > 0) - ((y0-y1) < 0);

            p = posLCtoM(x1-s1,y1-s2); /* Selection du pion pris */

            /* Pas de verif du premier pion au cas où il revient */
            move(p0,p,0);
            res2 = res2&&(detection()==0); /* Verification des prises */
            move(p,p1,0);
            if (j==n) { /* Deux pions ayant la même liste de prise ne peuvent arriver au même
endroit */
               res2 = res2&&(detection()==player);
```

```
            }

            p0 = p1;
            j++;
         }
      }
      i++;
   }

   if (matlab.writeable()) { /* Envoi de la position joué, Matlab attend carte */
      matlab.printf("%d",i); /* le +1 sert pour l'indexation matlab */
   }

   free(tab);
   move(p,51,0); /* Remise à zéro */
}
```