Rapport du projet de théorie des jeux

Maxence Ahlouche Maxime Arthaud Korantin Auguste
Martin Carton Thomas Forgione Thomas Wagner

11 novembre 2013

Table des matières

1 Présentation de l'équipe

Cette équipe a été menée par Thomas Forgione, assisté de son Responsable Qualité Maxence Ahlouche. Les autres membres de l'équipe sont Martin Carton, Thomas Wagner, Maxime Arthaud, et Korantin Auguste.

Tous les membres de l'équipe ont été présents à chacune des séances lors de cette UA.

2 Shifumi

Une stratégie simple et efficace à laquelle on pourrait penser pour gagner au Shifumi serait de jouer de manière aléatoire.

Et en effet, il s'avère que si les deux joueurs jouent de manière équiprobable, on a affaire à un équilibre de Nash : aucun changement de stratégie de la part d'un joueur ne pourra lui permettre d'augmenter ses chances de gagner.

De plus, si un adversaire ne joue pas de manière aléatoire (ou augmente les probabilités de jouer un certain élément), alors on pourra prévoir ce qu'il va jouer et donc trouver une stratégie qui pourra le battre. Les humains étant très mauvais pour jouer de manière aléatoire, il est assez facile d'écrire une stratégie permettant de les battre.

2.1 Stratégie développée

Afin de démontrer qu'un adversaire ne jouant pas aléatoirement est facile à battre, nous avons développé une stratégie qui se base sur des chaînes de Markov : en se basant sur les derniers éléments joués, elle regarde dans l'historique pour voir l'élément qui était joué le plus souvent par l'adversaire après les derniers coups joués.

Cette stratégie s'avère vraiment efficace contre un joueur humain. Toutefois, elle est prévisible : si on sait qu'on a affaire à une telle stratégie, on peut jouer de manière à la battre.

C'est pour cela qu'une stratégie aléatoire est la seule pouvant maximiser nos gains dans le pire des cas.

2.2 Variantes

Toutes les variantes du Shifumi qui consistent à rajouter des éléments pour obtenir un nombre d'éléments pair (par exemple pierre/papier/ciseaux/puits) vont créer un déséquilibre, car un élément sera moins efficace contre les autres. L'équilibre de Nash du jeu va alors consister à ne jamais jouer cet élément.

Si le nombre d'éléments est impair, alors le jeu pourra être équilibré, comme un Shifumi classique.

2.3 Tests

TODO

3 Jeu de la somme magique

3.1 Représentation sous forme de morpion

Ce jeu, comme expliqué dans les transparents présentés en cours, consiste à choisir, à tour de rôle, n nombres parmi n^2 afin que leur somme soit égale à $\frac{n(n^2+1)}{2}$.

Une représentation possible de ce jeu est le carré magique : les joueurs doivent choisir, l'un après l'autre une case dans un carré magique, leur but étant de contrôler une ligne, une colonne ou une diagonale entière du carré magique ; alors, les nombres qu'ils auront choisis totaliseront. De même, ce problème correspond exactement au jeu du morpion, étendu à des grilles $n \times n$.

Ainsi, une des stratégies possibles pour un joueur du jeu de la somme magique est de construire un carré magique, et de représenter les nombres choisis par l'adversaire par un rond dans la case correspondante. Afin de choisir un nombre, il suffit d'appliquer la stratégie de morpion de son choix sur le carré magique, et de retourner le nombre correspondant.

Le choix du carré magique n'importe pas. En effet, dans un carré magique sont présentes toutes les possibilités de combinaison de nombres pour obtenir la somme voulue. Par conséquent, peu importe le carré magique que l'on choisit, les représentations sous forme de morpion seront toutes équivalentes.

Le morpion étant un jeu où l'on essaie de minimiser la perte maximum, on peut s'intéresser à l'algorithme du minmax, pour déterminer une stratégie non-perdante.

3.2 Algorithme du minmax appliqué au jeu du morpion

L'algorithme du minmax consiste à évaluer toutes les positions de jeu atteignables depuis la position courante, sur une certaine profondeur (autrement dit, un certain nombre de tours de jeu), et à jouer de manière à atteindre la position la plus avantageuse, en supposant que l'adversaire joue toujours le meilleur coup pour lui-même (ce coup étant évalué avec notre propre fonction d'évaluation, qui n'est pas forcément la même que celle de l'adversaire).

Par conséquent, afin d'implémenter l'algorithme du minmax, il faut commencer par déterminer une fonction d'évaluation.

3.2.1 Fonction d'évaluation

La fonction d'évaluation que nous avons choisie est très simple : une ligne, colonne ou diagonale (que nous appelleront désormais simplement "ligne") complétée avec notre symbole (ce qui signifie qu'on a gagné) vaut $+\infty$; si, au contraire, l'adversaire a complété une ligne, alors cette ligne vaut $-\infty$. Une ligne contenant uniquement notre symbole rapporte le nombre d'occurrences de notre symbole dans cette ligne ; à l'inverse, une ligne contenant uniquement le symbole de l'adversaire rapportera négativement le nombre d'occurrences de ce symbole dans cette ligne. Toutes les autres lignes ne rapportent aucun point. Ainsi, l'évaluation d'une position de jeu est la somme des points rapportés par chacune de ses lignes, colonnes et diagonales.

3.2.2 Minmax

L'algorithme du minmax va construire (implicitement) l'arbre des coups possibles à partir de la position courante. L'évaluation d'un nœud de cet arbre sera :

— Si c'est au tour du joueur courant de jouer, le maximum de l'évaluation de ses fils ;
— Si c'est à l'adversaire de jouer, le minimum de l'évaluation de ses fils (i.e. on suppose que l'adversaire joue le meilleur coup à sa disposition, selon la fonction d'évaluation du joueur courant).

L'évaluation d'une feuille de l'arbre se fera par la fonction d'évaluation définie précédemment.

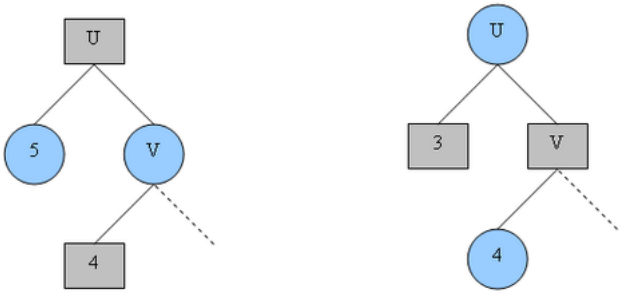
Ainsi, l'algorithme se dirigera naturellement vers la position la plus avantageuse pour lui.

3.3 Élagage alpha-beta

L'élagage alpha-beta permet de réduire le nombre de noeuds à parcourir durant l'algorithme du minmax.

Cette algorithme arrête le parcours des fils d'un noeud quand il se rend compte qu'il ne pourra pas faire mieux.

Dans l'exemple suivant, où les noeuds en bleus sont ceux où l'on doit prendre le minimum, et ceux en gris le maximum :



On se rend bien compte qu'on n'a pas besoin de parcourir les noeuds suivant, car on prend le maximum des minimums, ou l'inverse.

Par exemple, pour la coupure alpha : si on trouve des fils de V plus petit que 4, on va prendre le maximum, donc c'est 4 qui sera utilisé, et si on trouve plus grand ou égal à 5, on devra prendre le minimum au niveau de V, donc on prendra 5 dans tous les cas.

Au final, cette amélioration permet de gagner un temps considérable.

3.4 Résultats

3.4.1 Meilleure stratégie

Nous avons fait s'affronter différentes stratégies les unes contre les autres, afin de voir laquelle était la meilleure, sur différentes tailles de plateaux. Les stratégies que nous

avons fait s'affronter étaient :

— la stratégie aléatoire,
— la stratégie qui prend toujours la première case disponible,
— les stratégies utilisant le minmax à différentes profondeurs, entre 2 et 10,
— les stratégies utilisant le minmax et l'élagage alpha-beta à différentes profondeurs, entre 2 et 10.

Les résultats ne sont guère surprenants : toutes les stratégies parviennent au match nul, à l'exception des stratégies aléatoires et premier_dispo, qui perdent systématiquement contre un minmax de profondeur supérieure à 2.

3.4.2 Performances

Du point de vue du temps d'exécution, toutefois, l'algorithme avec élagage est beaucoup plus rapide qu'un minmax simple, alors qu'il retourne le même résultat. Malheureusement, nous n'avons pas pu tester nos algorithmes sur des plateaux de grande taille : en effet, à partir de la taille 4, un minmax avec élagage de profondeur 10 met en moyenne 5 secondes pour décider de son coup, et peut aller jusqu'à 40s !

Il est également intéressant de noter, bien que ceci n'ait rien à voir à notre projet, que le passage du langage Python au langage C a permis de diviser le temps d'exécution des tests par 60 pour des stratégies sans élagage.

4 Compétition/Duopole

Le but de ce jeu est de maximiser le gain d'une entreprise en concurrence avec une autre entreprise en fonction de leur production.

4.1 Analyse

Notre gain étant égal à $g_x(x) = -x(x + y - 3)$ avec x et y les productions respectives des deux entreprises, pour le maximiser il suffirait de jouer $x = \frac{3-y}{2}$ (c'est que fait la stratégie fournie `noncooperatif`).

Cependant au moment de décider quelle quantité produire nous ne connaissons pas la production y de l'entreprise concurrente. De plus si les deux entreprises s'ignorent totalement en essayant de maximiser leur seul gain, elles gagneront au final moins que deux entreprises qui coopèrent totalement c'est à dire qui chercheraient à maximiser leur gain total $g_x(x) + g_y(y)$ avec $g_x(x) = g_y(y)$ c'est à dire en jouant $x = y = 0.75$. En effet, elles gagneront alors chacune $\frac{9}{8}$ à chaque tour au lieu de 1 si elles sont toutes les deux non-coopératives.

Nous avons donc essayé plusieurs stratégies différentes, qui essayent de maximiser le gain de l'entreprise en tenant en compte l'autre entreprise, les productions et les gains des tours précédents.

4.2 Stratégies

4.2.1 Stackelberg

Une variante de la stratégie Stackelberg consiste à utiliser la production moyenne de l'autre joueur plutôt que seulement la dernière valeur. Elle permet d'obtenir des résultats légèrement meilleurs.

De plus en coopérant avec l'autre joueur (voir listing 1) si celui-ci coopère, on obtient de meilleurs résultats.

Enfin, une autre variante de cette stratégie (voir listing 2) maximise le gain si l'autre joueur a joué une constante sur les derniers tours. Cette variante donne des résultats un peu moins bons en moyenne mais est meilleure dans le meilleur des cas.

4.2.2 Stratégie pénalisante

Le principe de cette stratégie (voir listing 3) est d'être coopératif tant que l'adversaire l'est, et de devenir plus agressif quand il ne l'est plus : à chaque fois que l'autre joueur n'est pas coopératif, on joue comme le ferait la stratégie Stackelberg.

Une variante de cette stratégie (voir listing 4) consiste à le pénaliser de plus en plus : la première fois on le pénalise une fois, puis deux, puis trois, etc.

Ces deux stratégies sont efficaces à la fois quand l'autre joueur est coopératif (on est alors coopératif) et contre un joueur non-coopératif (on devient alors agressif).

4.2.3 Stratégie évolutive

Une autre stratégie (voir listing 5) que nous avons développée consiste à augmenter la production si la dernière augmentation a augmenté notre gain ou si la dernière diminution l'a diminué et vice-versa.

Celle-ci est plutôt efficace, mais n'est pas la meilleure que nous ayons développée : elle se met souvent à osciller inutilement.

4.2.4 Stratégie polynomiale

Enfin, une autre stratégie (voir listing 6) joue en fonction de la production moyenne de l'autre joueur telle que :

— $f(0) = 1.125$: on joue beaucoup si l'autre joue peu, sans jouer trop pour ne pas le fâcher ;
— $f(0.75) = 0.75$: elle coopère avec quelqu'un qui coopère ;
— $f(1.5) = 0.75$: elle coopère avec quelqu'un qui ne coopère pas, pour essayer de faire coopérer celui-ci (c'est dans notre intérêt et ça ne changerait pas son gain, il est donc possible qu'elle le fasse).

On choisit alors la fonction f pour être un polynôme qui interpole ces valeurs.

Cette méthode s'avère très efficace en moyenne.

4.3 Comparaison

Les tables 1 et 2 montrent les résultats obtenus par les quelques stratégies que nous avions à notre disposition [1] pour une durée de 99 tours et 1000 tours respectivement. Nous faisons s'opposer toutes les stratégies entre elles puis notons pour chacune d'elles son gain minimal, moyen et maximal [2].

On remarque que notre meilleure stratégie en moyenne est la stratégie qui utilise un polynôme (voir section 4.2.4), cette stratégie est aussi plutôt bonne dans le pire des cas (elle est donc adaptée à une entreprise qui souhaiterait minimiser ses risques).

Certaines stratégies peuvent prendre beaucoup de temps avant de devenir bonnes : c'est le cas par exemple de la stratégie « palkeo » qui est mauvaise dans le pire des cas sur 100 tours puis très bonne sur 1000 tours.

On remarque aussi que nos stratégie gagneront toujours de l'argent, contrairement à la stratégie nommée « killer » fournie par un autre groupe.

Enfin on remarque que la stratégie non-coopérative est plutôt bonne dans le pire des cas et en moyenne et est la meilleure dans le meilleur des cas. Elle est donc adaptée à une entreprise qui serait prête à prendre quelques risques pour avoir une chance de gagner plus.

5 Annexes

Listings

Listing 1 – Statégie Stackelberg sur la moyenne

```
% Stackelberg en moyenne, sauf si l'adversaire est coopératif.
function x = strategie(numpart,tx,ty,gx,gy)
if (numpart == 2)
        x= 0.75;
else
    ty_near_mean = mean(ty(max(numpart-5, 2):numpart-1));

    if (ty_near_mean < 0.76)
        x = 0.75;
    else
        ty_mean = mean(ty(2:numpart-1));
        x = 2*(3-ty_mean)/3;
    end;
end;
```

1. Les stratégies fournies par les professeurs (marquées *), les notre et des stratégies « prêtées » par d'autres groupes pour les tests (marquées **) que nous remercions.
2. Cette table peut être générée par le script matlab *comp_tests.m* fourni dans l'archive.

Listing 2 – Statégie Stackelberg sur la moyenne (variante)

```
% Stackelberg en moyenne, sauf si l'adversaire est coopératif avec
% maximisation des gains si l'adversaire est constant.
function x = strategie(numpart,tx,ty,gx,gy)
if (numpart == 2)
        x= 0.75;
else
    far = ty(max(2:numpart-15):max(2,numpart-1));
    cst = false;
    if (far == mean(far))
      cst = true;
    end;

    if (cst && numpart > 5)
        x = (3-mean(far))/2;
    else
        ty_near_mean = mean(ty(max(numpart-5, 2):numpart-1));

        if (ty_near_mean < 0.76)
            x = 0.75;
        else
            ty_mean = mean(ty(2:numpart-1));
            x = 2*(3-ty_mean)/3;
        end;
    end;
end;
```

Listing 3 – Statégie pénalisante

```
% Stratégie qui "pénalise" l'adversaire s'il n'est pas coopératif.
function x = strategie(numpart,tx,ty,gx,gy)
nbr_penal_y = 0;
nbr_penal_x = 0;

for i = 1:numpart-1
  if (ty(i) > 0.75)
    nbr_penal_y = nbr_penal_y + 1;
  end;
  if (tx(i) > 0.75)
    nbr_penal_x = nbr_penal_x + 1;
  end;
end;

if (nbr_penal_x <= nbr_penal_y)
  ty_mean = ty(2);
  if numpart > 2
    ty_mean = mean(ty(2:numpart-1));
  end;

  x = (3-ty_mean)/2;
else
  x = 0.75;
end;
```

Listing 4 – Statégie pénalisante (variante)

```
% Stratégie qui "pénalise" de plus en plus l'adversaire s'il n'est pas
% coopératif.
function x = strategie(numpart,tx,ty,gx,gy)
nbr_penal_y = 0;
nbr_penal_x = 0;
```

```
for i = 2:numpart-1
  if (ty(i) > 0.75)
    nbr_penal_y = nbr_penal_y + 1;
  end;
end;

while numpart-nbr_penal_x-1>0 && (tx(numpart-nbr_penal_x-1) > 0.75),
  nbr_penal_x = nbr_penal_x + 1;
end;

if (nbr_penal_x < nbr_penal_y)
  x = 1.1255*2*(3-ty(numpart-1))/3;
else
  x = 0.75;
end;
```

Listing 5 – Stratégie évolutive

```
function x = strategie(numpart,tx,ty,gx,gy)
persistent step;
if (numpart <= 2)
        x = 0.75;
        step = 1.3;
elseif (numpart == 3)
        x = 0.85;
else

        if ((gx(numpart-1)>gx(numpart-2)) && (tx(numpart-1)>tx(numpart-2)))
                x = tx(numpart-1) + step;
                step = step * 1.2;
        elseif (gx(numpart-1)<gx(numpart-2) && (tx(numpart-1)>tx(numpart-2)))
                x = tx(numpart-1) - step;
                step = step / 1.2;
        elseif  (gx(numpart-1)>gx(numpart-2) && (tx(numpart-1)<tx(numpart-2)))
                x = tx(numpart-1) - step;
                step = step * 1.2;
        else
                x = tx(numpart-1) + step;
                step = step / 1.2;
        end

        x = max(0.75, min(1.5, x));
end
```

Listing 6 – Stratégie polynomiale

```
% Stratégie qui joue telle que:
%   f(0)    = 1.125   (on est pas trop brutal)
%   f(0.75) = 0.75    (on coopère avec qqn qui coopère)
%   f(1.5)  = 0.75    (on essaye de coopérer avec qqn qui ne coopère pas, des
%                      fois qu'il changerait d'avis)
% en fonction de mean(ty(2:numpart-1)).
function x = strategie(numpart,tx,ty,gx,gy)
if (numpart <= 2)
    x= 0.75;
else
    x= 1/3*mean(ty(2:numpart-1))^2-3/4*mean(ty(2:numpart-1))+1.125;
end;
```

Stratégie	Gain minimal	Gain moyen	Gain maximum
cooperatif*	55.31	97.84	111.38
noncooperatif*	42.96	87.74	130.26
stackelberg*	48.00	79.75	114.69
palkeo	15.75	92.49	111.38
Pénalise	45.53	86.45	111.87
Pénalise (variante)	41.87	89.08	111.38
Stackelberg en moyenne	48.13	91.19	111.38
Stackelberg en moyenne variante	48.92	78.55	123.19
gklmjbse	55.31	83.89	116.29
poly	55.53	100.21	111.38
killer**	0.00	71.91	114.64
cooperatifmixte**	61.43	97.84	111.38
agressivemieux**	3.15	74.76	112.18
best_strategie**	32.11	86.15	122.62

TABLE 1 – Résultats des différentes stratégies sur 100 tours

Stratégie	Gain minimal	Gain moyen	Gain maximum
cooperatif*	561.56	978.59	1123.88
noncooperatif*	380.79	877.10	1317.08
stackelberg*	498.00	797.42	1132.44
palkeo	694.54	986.94	1123.88
Pénalise	419.12	860.12	1124.70
Pénalise variante	421.22	896.10	1123.88
Stackelberg en moyenne	492.11	923.94	1123.88
Stackelberg en moyenne (variante)	531.85	800.61	1262.25
gklmjbse	561.56	832.41	1135.33
poly	561.82	1011.24	1123.88
killer**	0.00	773.86	1133.09
cooperatifmixte**	698.67	990.58	1123.88
agressivemieux**	3.15	750.64	1126.18
best_strategie**	322.67	881.00	1262.81

TABLE 2 – Résultats des différentes stratégies sur 1000 tours