Projet MADI
Algorithmes pour la planification dans le risque

Manon SEPPECHER - Dan MIMOUNI

January 7, 2018

1 Générateur d'instances et visualisation

Notre générateur d'instances s'appuie sur les éléments de code proposés. Mais nous avons choisi de s'appuyer uniquement sur le second modèle, où la grille g est de dimension 3 : l'abscisse, l'ordonnée, et un troisième vecteur dont la première coordonnée correspond à la couleur et la deuxième au cout. En effet, les instances de la partie 2 ne sont que des cas particuliers de la partie 3, dans lesquels les couts pour une couleur donnée sont tous identiques.

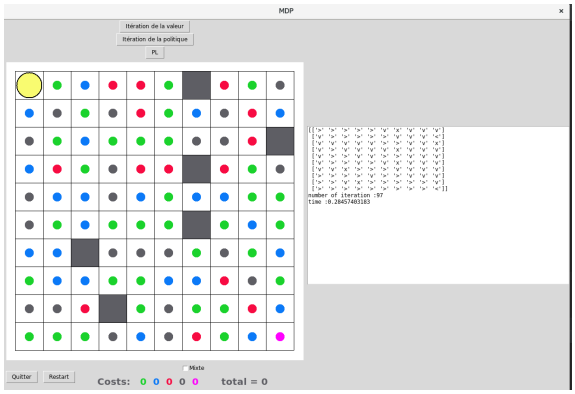

Figure 1: Grille sans chiffre

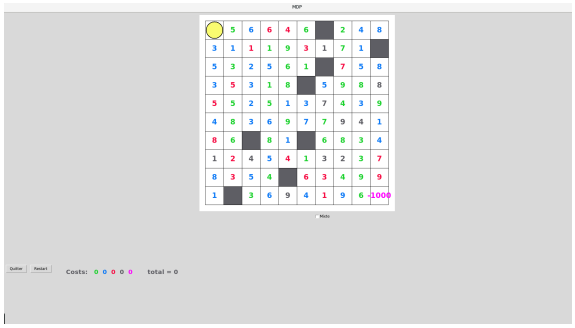

Figure 2: Grille avec chiffre

2 Recherche d'une trajectoire de moindre risque

2.1 Modélisation du problème comme PDM & équations de Bellmann

Dans cette partie, on s'intéresse à la recherche d'une trajectoire de moindre risque. On ne prend pas en compte les chiffres de la grille et l'on ne considère que les couleurs associées à chaque case. Chaque couleur représente un risque plus ou moins élevé, et on cherche à déterminer les politiques pour atteindre la case désirée dont l'espérance de coût est minimale.

Ceci peut se modéliser comme un processus décisionnel markovien S, A, T, R où :

- S : l'ensemble fini des états du monde. Correspond à chaque case de la grille accessible pour le robot, c'est-à-dire toutes les cases exceptées celles où il y a un mur
- A : l'ensemble fini des actions. Correspond aux quatre actions monter, descendre, aller à droite et aller à gauche
- T : la fonction de transition est telle que décrite en introduction :

  > "Par exemple apres un mouvement $\downarrow$ en direction d'une case x, cette case est atteinte avec la probabilite $p > 0,5$ mais le robot peut se retrouver dans la case située à gauche de x avec une probabilite $1-p/2$ ou à droite de x avec une probabilite $1-p/2$. Si l'une des deux cases (gauche ou droite) n'est pas accessible (case grisée) alors la case x est atteinte avec une probabilite $1+p/2$ et la seule case voisine accessible est atteinte avec un probabilite $1-p/2$. Si la case x n'est pas accessible alors on reste sur place avec une probabilite 1. Les consequences des autres actions s'en deduisent par symetrie."

- R : la récompense qui est en fait un coût et est définie par le tableau ci-dessous. Cette récompense est négative puisqu'elle représente en fait un coup.

x	vert	bleu	rouge	noir
r(x)	-1	-2	-3	-4

Il existe également une récompense immédiate pour la case cible qui est de 1000.

On peut donc écrire les équations de Bellman sous la forme :

$$V^*(s) = \max_a [R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')], \;\; s \in S$$

$R(s,a)$ ne dépendant que de l'état s courant, on obtient :

$$V^*(s) = \max_a [R(s) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')], \;\; s \in S$$

avec A, T et S définis plus haut. Dans les faits, il ne sera pas nécessaire de sommer les termes $T(s,a,s')V^*(s')$ sur tout s' mais seulement sur les trois cases potentiellement accessibles depuis notre position lorsque l'on vise une case voisine.

2.2 Résolution par itération de la valeur, itération de la politique, et programmation dynamique et influence des paramètres

L'algorithme d'itération de la valeur consiste a amélioré la valeur de chaque état à chaque itération. On s'arrête lorsque l'amélioration par rapport à l'état précédant est inférieur à epsilon. On déduit la politique optimal à partir des valeurs optimales des états

L'algorithme d'itération de la politique se déroule comme suit :

- On choisit une politique de départ aléatoire et on l'évalue
- On change cette politique et on l'évalue de nouveau
- On compare les deux et ainsi de suite jusqu'à que l'amélioration soit inférieur à epsilon

La programmation mathématique consiste en la minimisation d'un fonction objective qui est la somme de la valeur des états. Ce qui revient a minimisé la valeur des états. Il faut minimiser cette fonction avec pour contrainte que la valeur V(s) d'un état s soit supérieur à la valeur V(s) modifiée. Ensuite on en déduit la politique optimale.

Les différents algorithmes de résolution sont implémentés dans le module algo.py du projet.

Les tests ont été effectués pour des probabilités de 1, puis de 0.6.

On commence maintenant par s'intéresser à l'impact de la taille de l'instance sur les performances en temps et en nombre d'itération des algorithmes.

On obtient le dataframe suivant, que l'on représente sur les courbes ci-dessous. Les points bleus représentent l'itération de la valeur, les points oranges l'itération de la politique, et les points verts la résolution par programmation mathématique.

On observe qu'en nombre d'itération, les algorithmes d'itération de la valeur et de la politique ont une performance comparable. La résolution mathématique, elle, est bien moins performante. En terme de durée d'execution en revanche, c'est la méthode de programmation linéaire qui est la plus performante, alors que l'algorithme d'itération de la valeur arrive second et l'algorithme d'itération de la politique dernier.

dimensions-x	dimensions-y	taille	iter_val	iter_pol	iter_pl	time_val	time_pol	time_pl
5	5	25	93	94	24.33	$6.29 \cdot 10^{-2}$	$8.03 \cdot 10^{-2}$	$4.8 \cdot 10^{-3}$
5	10	50	94	95	46.6	0.13	0.17	$9.22 \cdot 10^{-3}$
10	10	100	97	98	91.8	0.27	0.33	$1.76 \cdot 10^{-2}$
10	15	150	99	100	150.87	0.42	0.51	$2.61 \cdot 10^{-2}$
15	15	225	102	103	223.67	0.63	0.79	$3.99 \cdot 10^{-2}$
15	20	300	105	106	302.47	0.86	1.08	$5.2 \cdot 10^{-2}$
20	20	400	107	108	402.07	1.18	1.49	$7.06 \cdot 10^{-2}$

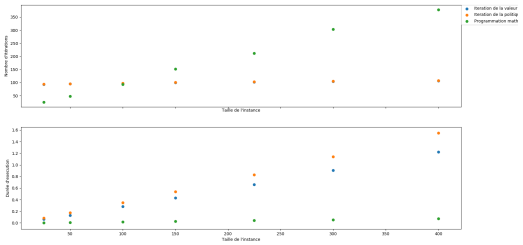

Figure 3: Nombre d'itérations et durée d'éxécution des algorithmes en fonction de la taille de l'instance

On s'intéresse également à l'impact du facteur d'actualisation gamma sur les performances des algorithmes. On observe qu'en terme de nombre d'itération, les résultats sont identiques pour l'algorithme d'itération de la valeur et celui d'itération de la politique. La résolution mathématique est bien moins performante à partir d'un gamma valant 0.4, mais devient à nouveau meilleure que les deux autres algorithmes pour une valeur élevée de gamma. En terme de durée d'éxécution, les trois algorithmes sont relativement aussi performants jusqu'à gamma = 0.3, puis la résolution par PL devient très avantageuse, devant celle par itération de la valeur, puis celle par itération de la

politique.

gamma	iter_val	iter_pol	iter_pl	time_val	time_pol	time_pl
0	2	2	0	$5.79 \cdot 10^{-3}$	$6.85 \cdot 10^{-3}$	$1.93 \cdot 10^{-2}$
0.1	7	8	4.47	$2.04 \cdot 10^{-2}$	$2.86 \cdot 10^{-2}$	$1.96 \cdot 10^{-2}$
0.2	10	11	8.47	$2.81 \cdot 10^{-2}$	$3.73 \cdot 10^{-2}$	$1.97 \cdot 10^{-2}$
0.3	12	13	12.73	$3.36 \cdot 10^{-2}$	$4.53 \cdot 10^{-2}$	$1.97 \cdot 10^{-2}$
0.4	15	16	22.6	$4.31 \cdot 10^{-2}$	$5.71 \cdot 10^{-2}$	$1.9 \cdot 10^{-2}$
0.5	20	21	36.33	$5.41 \cdot 10^{-2}$	$6.97 \cdot 10^{-2}$	$1.86 \cdot 10^{-2}$
0.6	26	27	59.6	$7.2 \cdot 10^{-2}$	$8.85 \cdot 10^{-2}$	$1.97 \cdot 10^{-2}$
0.7	34	35	85.87	$9.06 \cdot 10^{-2}$	0.12	$1.98 \cdot 10^{-2}$
0.8	52	53	100.73	0.14	0.18	$1.88 \cdot 10^{-2}$
0.9	96	97	86.6	0.26	0.32	$1.86 \cdot 10^{-2}$

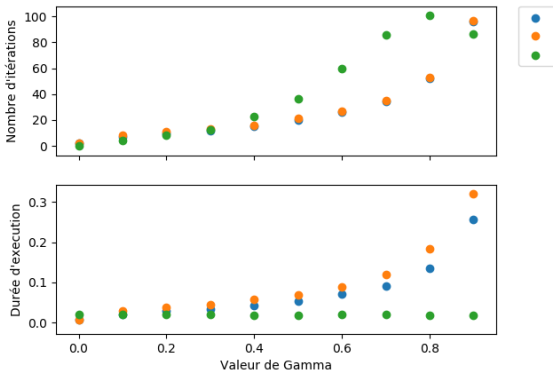

Figure 4: Nombre d'itérations et durée d'éxécution des algorithmes en fonction de gamma

Enfin, on décide d'étudier l'impact de la probabilité p sur les performances de ces trois algorithmes.

p	iter_val	iter_pol	iter_pl	time_val	time_pol	time_pl
0	93	95	113.8	0.29	0.37	$2.8 \cdot 10^{-2}$
0.1	93	95	116.27	0.3	0.38	$3.06 \cdot 10^{-2}$
0.2	98	99	114.4	0.31	0.39	$3.1 \cdot 10^{-2}$
0.3	97	98	117.2	0.31	0.39	$3.04 \cdot 10^{-2}$
0.4	98	99	119.53	0.31	0.39	$3.09 \cdot 10^{-2}$
0.5	96	97	115.73	0.31	0.39	$3.08 \cdot 10^{-2}$
0.6	96	97	102.07	0.31	0.38	$3.09 \cdot 10^{-2}$
0.7	99	100	120.67	0.32	0.4	$3.12 \cdot 10^{-2}$
0.8	97	98	116.87	0.31	0.39	$3.04 \cdot 10^{-2}$
0.9	98	99	118.13	0.31	0.39	$3.1 \cdot 10^{-2}$

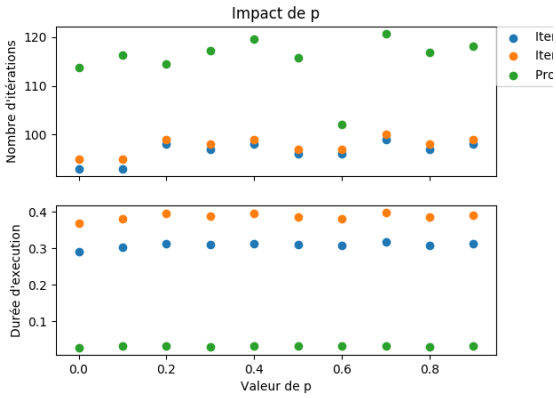

Figure 5: Nombre d'itérations et durée d'éxécution des algorithmes en fonction de p

2.3 Variation de la solution du problème en fonction de q

On va maintenant remplacer la fonction coût $c(x)$ par la fonction $c^q(x)$ ou q > 1. Par exemple pour q $= 2$, on aura les couts :

x	vert	bleu	rouge	noir
$c^2(x)$	1	4	9	16

En faisant varier q de façon croissante, on s'aperçoit que les actions changent au fur et à mesure. Alors que la case cible était accessible depuis l'origine pour q petit, elle est les autres cases eloignées ne correspondent plus à des politiques permettant la convergence vers la cible au fur et à mesure que q augmente. Ceci se conçoit facilement grâce à deux effets conjugués : le fait que les couts vont progressivement devenir équivalents ou supérieurs à la récompense finale, ce qui rend les déplacement inintéressant, et la topologie du terrain, où les cases de coûts élevés vont être évitées.

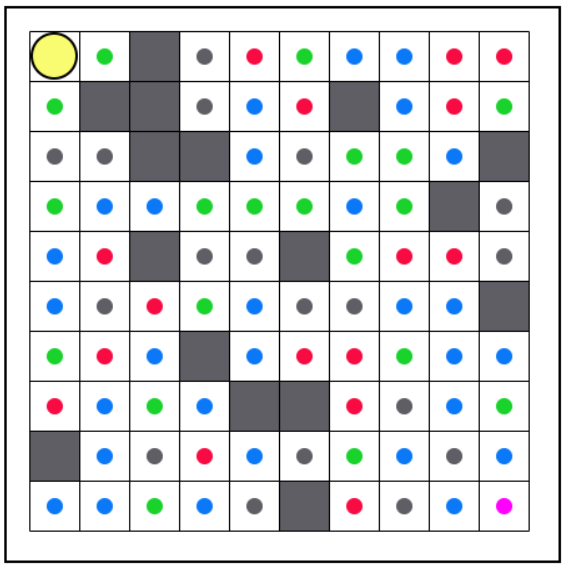

Figure 6: Grille de simulation de la variation de q

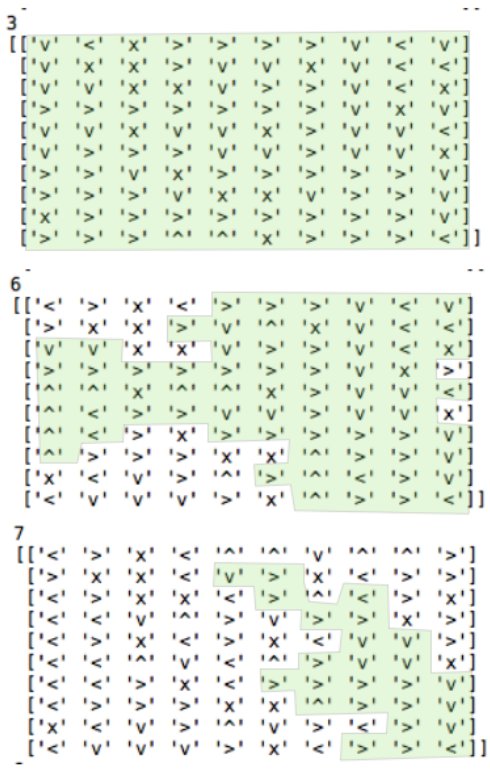

Figure 7: Evolution de la politique quand q croit

2.4 Changement de la méthode d'évaluation d'une trajectoire

On considère maintenant qu'une politique est meilleure qu'une autre si elle traverse moins de cases noires ou, en cas d'égalité, si elle traverse moins de cases rouges ou, en cas d'égalité sur les noires et les rouges, si elle traverse moins de cases bleues, ou bien encore, en cas d'égalité sur les bleues, si elle traverse moins de cases vertes.

On peut modéliser ce problème comme précédemment, en changeant les récompenses, de façon à chercher à minimiser en priorité le nombre de cases noires traversées, puis les rouges, puis les bleues et enfin les vertes.

Pour cela, il faut que le coût d'une case noire soit suffisamment élevé pour qu'une trajectoire au pire cas contenant uniquement des cases rouges (respectivement bleues, vertes) soit quand même considérée meilleure qu'une trajectoire contenant une seule case noire (respectivement rouge, bleue).

Ainsi, on doit avoir :

$$c(noir) > l \cdot c(rouge)$$
$$c(rouge) > l \cdot c(bleu)$$
$$c(bleu) > l \cdot c(vert)$$

avec l la longueur au pire cas de l'itinéraire du robot. Le robot étant positionné sur une première case, il lui en reste $n \cdot m - 1$ à explorer au pire cas. On peut donc écrire :

$$c(noir) \geq n \cdot m \cdot c(rouge)$$
$$c(rouge) \geq n \cdot m \cdot c(bleu)$$
$$c(bleu) \geq n \cdot m \cdot c(vert)$$

On propose alors la grille de coûts suivante :

x	vert	bleu	rouge	noir
c(x)	1	$n \cdot m$	$(n \cdot m)^2$	$(n \cdot m)^3$

En conservant une récompense de 1000 pour la case cible, on obtient la matrice de décision suivante : le pion n'atteint donc pas la cible. Ceci est compréhensible puisque les coûts sont très importants relativement à la récompense. Si l'on augmente la récompense, on obtient au contraire une cible accessible par depuis presque l'intégralité du terrain. Mais on observe que depuis certaines zones de coûts forts (les cases rouges et noires de la dernière colonnent), il n'est pas intéressant de chercher à atteindre la cible, puisque cela ferait traverser plus de cases noires que de rester sur place.

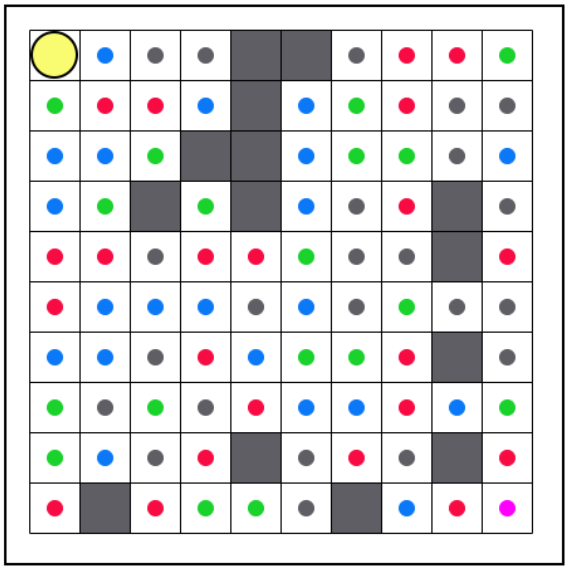

Figure 8: Grille utilisée pour les résultats suivants

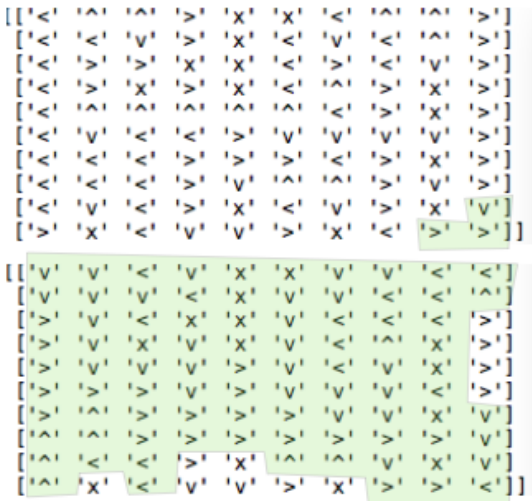

Figure 9: Zones dont les politiques peuvent permettre d'atteindre la cible, pour une récompense respectivement de 10000 et de 100000

3 Recherche d'une trajectoire équilibrée

Dans cette partie on considère qu'à chaque case est affecté un chiffre et une couleur. Les couleurs représentent des ressources, et les chiffres représentent la quantité de la ressource en question consommée lors du passage sur la case. On cherche une politique qui minimise la consommation des ressources.

3.1 Minimisation de l'espérance de la consommation totale des ressources

Dans un premier temps, on cherche à minimiser l'espérance de la consommation *totale* des ressources. Il n'est donc pas nécessaire de faire de distinction entre les différents types de ressources (ou couleurs) : il suffit de s'intéresser à la somme des quantités consommées, toutes ressources confondues, c'est à dire à la somme des chiffres des cases traversées.

Ceci peut se décrire comme un PDM de la manière suivante :

- S l'ensemble fini des états du monde correspond à chaque case de la grille accessible pour le robot, c'est-à-dire toutes les cases exceptées celles où il y a un mur ;
- A l'ensemble fini des actions correspond aux quatre actions monter, descendre, aller à droite et aller à gauche ;
- T la fonction de transition est telle que décrite en introduction ;
- enfin la récompense immédiate R est en fait l'opposé du coût en ressource associé à la case.

3.2 Critère min-max

Mais l'approche précédemment décrite, qui cherche à minimiser le coût ou consommation totale en ressource, ne garantie pas que les ressources soient consommées de façon équilibrées. C'est pourquoi l'on s'intéresse au critère :

$$\min_{\pi} \max_{c_i(\pi)}$$

Ici les méthodes classiques de résolution de MDPs ne sont pas adaptées car on est en présence de plusieurs objectifs distincts : à la fois celui de minimiser la quantité de ressources consommées, et en même temps de consommer ces ressources de façon équilibrées.
Ainsi, une solution qui consomme 10 unités d'une ressource A et 0 unité de ressources B, C, D sera jugée moins satisfaisante qu'une solution qui consomme 9 unités sur chacune des ressources, alors que le total des ressources consommées sera plus faible dans le premier cas que dans le second. En revanche une solution qui consomme 8 sur chaque ressource sera jugée plus satisfaisante.

C'est pour cela qu'un modèle MDP ne peut pas être satisfaisant et que l'on doit se placer dans le cadre des MOMDPs.

Dans un premier temps, notons que le critère min-max sur les couts est équivalent à un critère max-min sur les gains.

On considère donc les coûts en chacune des ressources sont des récompenses négatives.

On peut alors se ramener au PL :

Donne :

$$\max z$$

$$\begin{aligned} z &\leq f_i(x) && \forall i \in \{vert, bleu, rouge, noir\} \\ f_i(x) &= \sum_{s\in S}\sum_{a\in A} R_i(s,a)x_{sa} && \forall i \in \{vert, bleu, rouge, noir\} \end{aligned}$$

$$\begin{aligned} s.t. \quad \sum_{a\in A} x_{sa} - \gamma \sum_{s'\in S}\sum_{a\in A} T(s',a,s)x_{s'a} &= \mu(s) && \forall s \in S \\ x_{sa} &\geq 0 && \forall s \in S, \forall a \in A \end{aligned}$$

Si l'on ne s'intéresse qu'à des politiques pures, on ajoute les contraintes :

$$\begin{aligned} \sum_{a\in A} d_{sa} &\leq 1 && \forall s \in S \\ (1-\gamma)\cdot x_{sa} &\leq d_{sa} && \forall s \in S,\ \forall a \in A \\ d_{sa} &\in 0,1 && \forall s \in S,\ \forall a \in A \end{aligned}$$

Or $R_i(s,a) = R_i(s)$ car le coût induit par le fait de traverser la case ne dépend que de celle-ci et pas de l'action menée par la suite.

Donc:

$$\max z$$

$$\begin{aligned} z &\leq f_i(x) && \forall i \in \{vert, bleu, rouge, noir\} \\ f_i(x) &= -\sum_{s\in S}\sum_{a\in A} c_i(s)x_{sa} && \forall i \in \{vert, bleu, rouge, noir\} \end{aligned}$$

$$\begin{aligned} s.t. \quad \sum_{a\in A} x_{sa} - \gamma \sum_{s'\in S}\sum_{a\in A} T(s',a,s)x_{s'a} &= 1 && \forall s \in S \\ x_{sa} &\geq 0 && \forall s \in S, \forall a \in A \end{aligned}$$

Encore une fois, si l'on ne s'intéresse qu'à des politiques pures, on ajoute les contraintes :

$$\begin{aligned} \sum_{a\in A} d_{sa} &\leq 1 && \forall s \in S \\ (1-\gamma)\cdot x_{sa} &\leq d_{sa} && \forall s \in S,\ \forall a \in A \\ d_{sa} &\in \{0,1\} && \forall s \in S,\ \forall a \in A \end{aligned}$$

En divisant chaque x_{sa} par la somme de x_{sa}sur les actions possibles, on pourrait alors se ramener à une "stationary randomized policy", où pour un état donné chaque action est déterminé avec une certaine probabilité.

Les deux PL correspondants ont été entièrement implémentés, tout comme les fonctions permettant de répondre aux questions suivantes. Ces éléments de code sont consultables avec le reste du projet. Cependant, nous n'avons pas réussi à obtenir les résultats attendus.

4 Conclusion

En conclusion, nous avons pu implémenter sur deux problèmes relativement similaires les algorithmes standards de résolution de PDM. Nous avons pu comparer leurs performances et évaluer l'impact de leurs différents paramètres sur les résultats obtenus.

Pendant ce projet, nous avons envisagé des perspectives d'amélioration de notre travail. Nous avons pensé qu'il serait intéressant d'étudier les politiques pour des topologies particulières de grille, ou pour des positions d'origines et de cibles différentes.

Il serait également intéressant d'améliorer l'interface pour permettre de visualiser rapidement quelles sont les cases depuis lesquelles la cible est accessibles et quelles sont celles depuis lesquelles elle ne l'est pas.

Une autre étude intéressante serait de permettre des mouvements diagonaux et plus seulement horizontaux et verticaux. Cela pourrait introduire d'avantage de continuité dans les résultats et des affichages proches de ceux de champs de vecteurs.

Enfin, nous regrettons de n'avoir pas pu analyser les résultats de la partie 3, qui auraient certainement été très intéressants.