Rapport de Projet MADI

Processus de décision markoviens et apprentissage par renforcement

Alexandre Bontems & Gualtiero Mottola

Table des matières

1 Framework et solveur PDM

Dans ce projet on considère un jeu dans lequel un joueur doit se déplacer au sein d'un labyrinthe et doit remplir les objectifs suivants : trouver un trésor, trouver la clé qui permet d'y accéder et sortir du labyrinthe avec le trésor. Le joueur se déplace de salle en salle qui peuvent être de plusieurs types (voir table 1 qui détaille les événements pouvant survenir). Trois objets sont également récupérables : l'épée, la clé et le trésor. Lorsque le joueur possède l'épée, il est capable de vaincre les ennemis à coup sûr et pour récupérer le trésor il est nécessaire d'avoir la clé.

Type de salle	**Symbole**	**Description**
Position de départ	•	
Salle vide	(blank)	
Mur	■	Impossible de s'y déplacer.
Ennemi	E	Le joueur tue son adversaire avec probabilité 0.7 ou meurt.
Piège	▲	Le joueur peut mourir (avec probabilité 0.1), être déplacé à la position de départ (probabilité 0.3) ou rien ne se passe.
Fissure	C	Mort immédiate.
Trésor	T	
Épée	†	
Clé	K	
Portail	○	Le joueur est déplacé dans une salle aléatoire du labyrinthe.
Plateforme mouvante	-	Le joueur est aléatoirement déplacé sur une des cases voisines de la plateforme.

Table 1 – Description des différents types de salle.

L'objectif du projet est de déterminer la politique optimale de déplacement pour réussir le jeu. Pour cela on développe tout d'abord un framework permettant de manipuler le jeu et développer des algorithmes de résolution. Le logiciel développé doit être capable d'afficher des grilles de labyrinthes et de proposer à l'utilisateur un mode interactif pour jouer aux grilles. On veut aussi être capable de générer différentes grilles aléatoirement en spécifiant leurs caractéristiques.

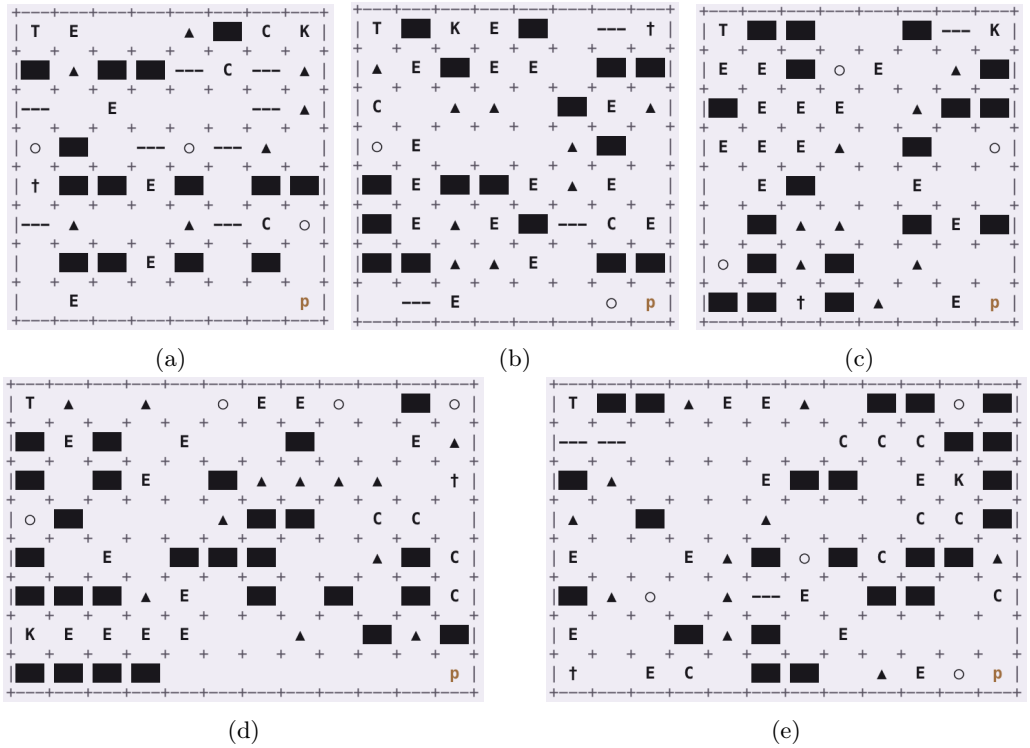


FIGURE 1

Toutes les instances considérées pour les évaluations expérimentales des méthodes de résolution sont visibles sur la figure 1.

1.1 Modélisation du PDM

Pour trouver les politiques optimales de déplacement dans ce jeu, on décide de modéliser le problème comme un Processus de Décision Markovien (PDM) dont les états correspondent à l'état du joueur à un moment donné. Soit $\mathcal{S}$ l'ensemble des états possibles, on définit un état du joueur comme la structure suivante :

$$(x,\, y,\, \texttt{has\_key},\, \texttt{has\_sword},\, \texttt{has\_treasure}) \tag{1}$$

où le couple de variables (x, y) décrit la position du joueur dans le labyrinthe et les variables booléennes `has_key`, `has_sword`, `has_treasure` et `is_dead` décrivent les objets que le joueur possède. On compte également deux états terminaux pour la victoire et la défaite (mort) du joueur.

À chaque pas de temps, le joueur peut effectuer une action parmi les options suivantes :

$$\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\} \tag{2}$$

Cependant si une direction mène à un mur, l'action correspondante n'est pas disponible. De plus, certains états ne permettent pas au joueur d'effectuer une action : si le joueur est dans un portail ou sur une plateforme par exemple, aucune action n'est possible et le joueur est déplacé selon les règles du jeu.

En ce qui concerne les transitions et les probabilités $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ associées, elles sont calculées en fonction de l'état d'arrivée. Si la salle atteinte après un mouvement est une case vide ou une salle contenant un objet, alors la probabilité d'atteindre l'état correspondant est de 1. Si la salle contient un piège alors, comme spécifié en table 1, on arrive dans l'état terminal de défaite avec probabilité 0.1, dans l'état de la salle de départ (mais avec les objets

courants) avec probabilité 0.3, ou dans la salle qui contient le piège avec probabilité 0.6. Si la salle contient une fissure alors on atteint l'état de défaite à coup sûr. Si la salle contient un ennemi alors l'état de défaite est atteint avec probabilité 0.3 et l'état de la salle avec probabilité 0.7. Enfin une action menant vers une salle plateforme ou portail correspond à une transition vers une des salles voisines de la plateforme ou une salle aléatoire dans la labyrinthe pour le portail.

La fonction récompense $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ renvoie une valeur réelle en fonction de l'état d'arrivée du joueur. Ainsi la valeur $R(s, a)$ dépend de l'état s' atteint après avoir effectué l'action a depuis l'état s et des différents objets que le joueur possède. On utilise les valeurs de la table 2 pour la suite. Pour favoriser la recherche du trésor en un nombre de mouvement minimum, on pénalise légèrement les mouvements inutiles. La récupération d'objets, quant à elle, est largement récompensée puisqu'elle ne peut qu'être bénéfique au joueur.

Type de salle	r
Position de départ	-1
Salle vide	-1
Mur	-100
Enemi	-1
Piège	-1
Fissure	-1
Trésor	-1
Épée	100
Clé	100
Portail	-1
Plateforme mouvante	-1
Avec clé :	
· Trésor	1 000
· Clé	-1
Avec épée :	
· Épée	-1
Avec trésor :	
· Position de départ	1 000 000
Joueur mort :	-100

Table 2 – Récompenses en fonction de l'état d'arrivée du joueur

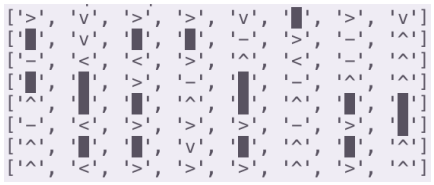
1.2 Optimisation par itération de la valeur

Pour l'implémentation de l'algorithme d'itération de la valeur on suit les étapes de l'algorithme 1. À chaque itération, la valeur de chacune des actions sur tous les états est réévaluée. On utilise pour cela un dictionnaire qui pour chaque position dans la grille stocke la valeur des actions possibles pour ce noeud. On note que dans cet algorithme un mouvement vers un mur ou vers une position qui n'est pas dans la grille est interdit, c'est-à-dire que dans certains états le nombre d'actions possible diminue.

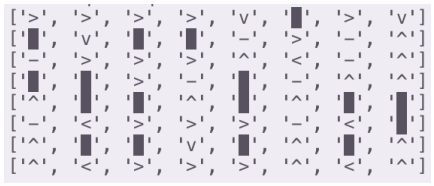
L'algorithme d'itération de la valeur est relancé plusieurs fois pour trouver la politique optimale dans chacun des états suivants définit par les variables suivantes : la santé du joueur est critique (un seul point de vie), le joueur a la clef, le joueur a le trésor et le joueur a l'épée. On ne lance cependant pas l'algorithme pour des états impossibles (ex : avoir le trésor mais pas la clef). Au total on calcule 12 politiques et on les applique ensuite en fonction de l'état du jeu.

Algorithme 1 Itération de la valeur.

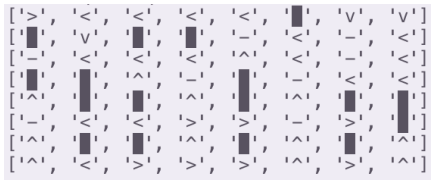
$V_0(s) \leftarrow 0 \quad \forall s \in \mathcal{S}$



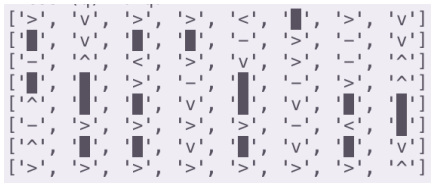
(a) Sans objets



(b) Avec épée



(c) Avec épée et clé



(d) Avec tous les objets

FIGURE 2 – Politique apprise pour le premier niveau par itération de la valeur.

$t \leftarrow 0$
tant que $\max_{s \in \mathcal{S}} \{|V_t(s) - V_{t-1}(s)|\} < \epsilon$ **faire**
  $t \leftarrow t + 1$
  pour $s \in \mathcal{S}$ **faire**
    pour $a \in \mathcal{A}$ **faire**
      $Q_t^a(s) \leftarrow R(s,a) + \gamma \sum_{s'} T(s,a,s') V_{t-1}(s')$
    fin pour
    $V_t(s) \leftarrow \max_{a \in \mathcal{A}} Q_t^a(s)$
  fin pour
fin tant que
pour $s \in \mathcal{S}$ **faire**
  $d(s) \leftarrow \text{choice}\left[\arg\max_{a \in \mathcal{A}} Q_t^a(s)\right]$
fin pour
Retourner d

Évaluation expérimentale La table 3 montre les résultats de l'algorithme d'itération de la valeur pour les instances considérées. Les paramètres suivants sont utilisés : un facteur d'actualisation γ de 0.5 et un ϵ de 10^{-14}.

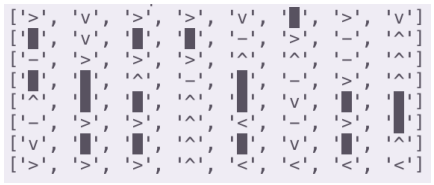
Instance	Nombre d'itérations	Temps d'exécution (secs)
lvl-n8-0 (a)	730	0.39
lvl-n8-1 (b)	731	0.38
lvl-n8-2 (c)	725	0.40
lvl-n12-1 (d)	749	0.58
lvl-n12-2 (e)	729	0.59

TABLE 3 – Performance de l'algorithme d'itération de la valeur.

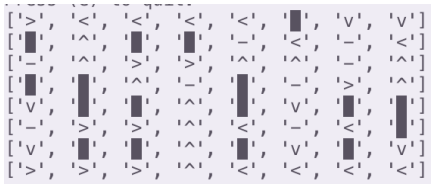
1.3 Optimisation par itération de la politique

On implémente également l'algorithme de l'itération de la politique. On utilise ici la librairie Numpy pour résoudre l'équation de l'algorithme. Comme pour l'itération de la valeur nous relançons ici l'algorithme plusieurs fois pour déterminer la politique optimale en fonctions des différents états du joueur.

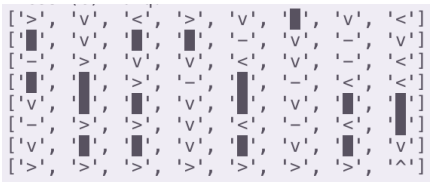
Algorithme 2 Itération de la politique.



(a) Sans objets



(b) Avec clé



(c) Avec clé et trésor

FIGURE 3 – Politique apprise pour le premier niveau par itération de la politique.

$d_0(s) \leftarrow$ politique aléatoire
$t \leftarrow 0$
tant que $d_t(s) \neq d_{t+1}(s) \forall s \in \mathcal{S}$ **faire**
  résoudre $\{V_t(s) = R(s, d_t(s)) + \gamma \sum_{s'} T(s, d_t(s), s') V_t(s'),\ \forall s \in \mathcal{S}\}$
  pour $s \in \mathcal{S}$ **faire**
    $d_{t+1}(s) \leftarrow \text{choice}\left[\arg\max_{a \in \mathcal{A}} \{R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_t(s')\}\right]$
  fin pour
  $t \leftarrow t + 1$
fin tant que
Retourner d_{t+1}

Évaluation expérimentale La table 4 montre les résultats de l'algorithme d'itération de la politique pour les instances considérées. Les paramètres suivants sont utilisés : un facteur d'actualisation γ de 0.99.

Instance	Nombre d'itérations	Temps d'exécution (secs)
lvl-n8-0 (a)	2100	2.65
lvl-n8-1 (b)	10069	11.38
lvl-n8-2 (c)	4091	4.70

TABLE 4 – Performance de l'algorithme d'itération de la valeur.

Les politiques obtenues sont similaires avec les deux méthodes d'itération de la valeur et d'itération de la politique. On note cependant que l'agent ne vas que très rarement chercher l'épée lorsque l'on utilise l'itération de la Politique. D'une manière générale les politiques trouvées ont tendance à éviter les portails. Cela est sans doute dû à la topographie des cartes utilisées car ce comportement n'est pas observé dans des cartes qui ne contiennent pas de salles létales.

1.4 Fonctionnalité supplémentaire

La possibilité pour le joueur d'avoir plus d'un point de vie a été ajoutée comme fonctionnalité supplémentaire. Pour refléter ce changement il a fallu étendre l'espace des états. En effet, lorsque le joueur a plus d'un point de vie au début du jeu, la probabilité d'atteindre l'état terminal de défaite n'est supérieure à 0 que lorsque le joueur a prit des dégats et se retrouve

avec un seul point de vie. On redéfinit donc les états du PDM comme suit :

$$s = (x, y, \texttt{has\_key}, \texttt{has\_sword}, \texttt{has\_treasure}, \texttt{critical}) \tag{3}$$

où le booléen `critical` indique si le joueur a un niveau critique de santé (c'est-à-dire un seul point de vie). Les algorithmes d'itération de la valeur et d'itération de la politique sont inchangés et utilisent les nouvelles valeurs de T.

2 Apprentissage par renforcement (Q-learning)

Après avoir résolu le PDM de manière exacte, on se place dans le cas où l'environnement est inconnu et donc les probabilités de transitions entre états sont inconnues. Un algorithme d'apprentissage par renforcement peut alors être utilisé pour approcher les valeurs de politique optimales V^*. L'idée est d'itérer des explorations du labyrinthe par le joueur pour apprendre les transitions et les récompenses associées aux différentes salles.

Dans l'algorithme Q-learning utilisé, le joueur parcourt le labyrinthe de salle en salle et met à jour la fonction Q selon la règle de mise à jour suivante :

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a') - Q_t(s_t, a_t) \right) \tag{4}$$

On y voit que l'algorithme est régit par deux paramètres α et γ qui représente respectivement le taux d'apprentissage et le facteur d'actualisation. Pour chaque état, on a également quatre actions possibles qui correspondent chacune à un mouvement dans la direction d'un des points cardinaux :

$$\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\} \tag{5}$$

La fonction récompense utilisée reprend les mêmes valeurs que dans les algorithmes précédents (table 2) et la valeur r_t dans (4) dépend donc uniquement de l'état s_{t+1} et de son état de santé.

L'implémentation se base sur le framework développé dans la partie précédente et propose donc l'apprentissage d'une politique optimale selon Q dans un labyrinthe donné. Pour cela on se base sur l'algorithme 3. On y voit apparaître un paramètre supplémentaire ϵ qui sert à éviter des optima locaux. En effet, durant la phase d'apprentissage, il est fréquent de voir le joueur osciller entre plusieurs salles sans pouvoir sortir d'une telle boucle. On utilise donc une stratégie ϵ-greedy pour pallier ce problème et s'assurer que chaque épisode ait un nombre d'étapes fini. Le joueur choisira donc, à chaque étape, soit l'action qui maximise Q soit une action aléatoire avec probabilité ϵ.

Algorithme 3 Apprentissage Q-learning.

$Q(s, a) \leftarrow$ valeur arbitraire $\forall (s, a)$
pour chaque épisode **faire**
  $s_0 \leftarrow (0, 0, \mathsf{F}, \mathsf{F}, \mathsf{F})$ ▷ Position de départ, pas d'objets.
  pour chaque étape t **faire**
    $p \leftarrow$ valeur aléatoire dans $[0, 1]$
    si $p \leq \epsilon$ **alors**
      $a_t \leftarrow$ action aléatoire dans $\mathcal{A}$
    sinon
      $a_t \leftarrow$ choisir $[\arg\max_a Q(s_t, a)]$
    fin si
    r_t, $s_{t+1} \leftarrow$ Faire action a_t.
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$
  fin pour
fin pour

Pour éviter que l'apprentissage ne soit trop influencé par ce paramètre, il est choisi grand pour les premiers épisodes et décroît jusqu'à 0 en fin d'apprentissage. Il est cependant possible de trouver des optima locaux pendant un épisode alors que ce paramètre a atteint sa valeur minimale. L'algorithme implémenté est alors capable d'augmenter sa valeur et de la faire décroître à nouveau pendant un petit nombre d'épisodes. Grâce à un ϵ grand dans les premiers épisodes et à des valeurs initiales Q élevées, on favorise l'exploration de l'environnement en début d'apprentissage.

L'exécution de l'algorithme utilise un nombre minimal d'itération et un nombre maximum d'itération. Dans cet intervalle, l'algorithme s'arrête s'il n'a pas eu à augmenter ϵ pendant un certain nombre d'itérations. On essaye ainsi d'éliminer les boucles infinies dans la politique apprise.

Évaluation expérimentale Pour tous les résultats présentés par la suite on utilise les valeurs de paramètres suivantes : un taux d'apprentissage α de 0.1, un facteur d'actualisation γ de 0.9 et un ϵ initial de 1. Chaque hausse de ϵ se fait à 0.1 et décroît ensuite linéairement avec le nombre d'itération. Le joueur possède également un seul point de vie de départ. L'algorithme effectue un nombre minimal d'itération fixé à 40 000 et un nombre maximum d'itération fixé à 80 000. Si ϵ n'est pas modifié pendant 10 000 itérations consécutives et que le nombre d'itérations minimum a été atteint, l'algorithme s'arrête.

Les politiques apprises pour le premier niveau (figure 1a) sont visibles dans la figure 5. On présente également les politiques pour le premier niveau de 12×10 (figure 1d) dans la figure 6. On remarque l'utilisation fréquente des portails pour se déplacer rapidement et l'utilisation intensives des pièges pour revenir à la sortie lorsque l'on a le trésor.

Instance	**Itérations**	**Victoires**	**Temps (secs)**
lvl-n8-0 (a)	40 000 (min)	3911 (9.8%)	68.42
lvl-n8-1 (b)	min	10 017 (25.0%)	45.01
lvl-n8-2 (c)	min	6427 (16.1%)	127.36
lvl-n12-1 (d)	min	7633 (19.1%)	108.20
lvl-n12-2 (e)	min	5027 (12.6%)	49.27

TABLE 5 – Performance de l'algorithme d'itération de la valeur.

On voit dans la figure 4 que l'exploration dure en général un nombre relativement petit d'itérations mais on laisse tourner l'algorithme au-delà pour s'assurer d'avoir éliminé toutes les boucles infinies qui pourraient survenir dans les politiques apprises. La table 5 reporte les performances de l'algorithme sur les instances considérées. On y voit que le nombre d'itérations minimum choisi est rarement dépassé. Le temps de résolution semble également être peu affecté par la taille de l'instance mais plus par la difficulté de l'instance. Puisque l'algorithme n'a pas besoin d'explorer tous les états possibles pour trouver la politique optimale, on comprend pourquoi la taille de l'instance influe peu sur le temps de résolution. Si le problème est facile, alors on est capable d'explorer plus d'état et l'algorithme met plus de temps à résoudre l'instance.

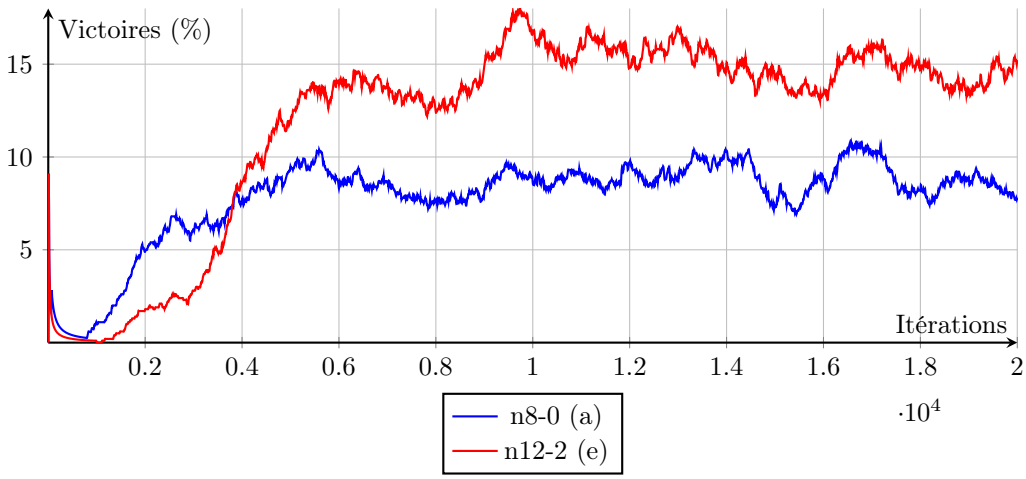
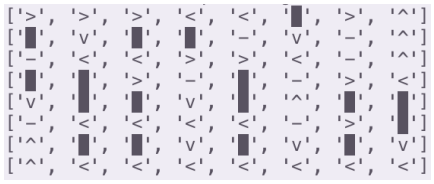
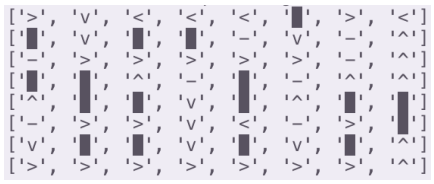


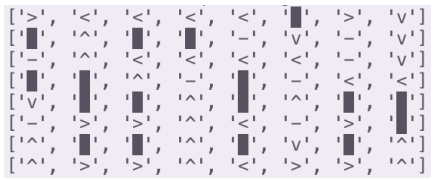
FIGURE 4 – Pourcentage de victoire sur les 1 000 dernières itérations.



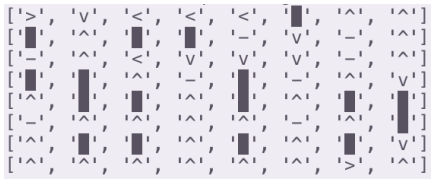
(a) Sans objets



(b) Avec épée

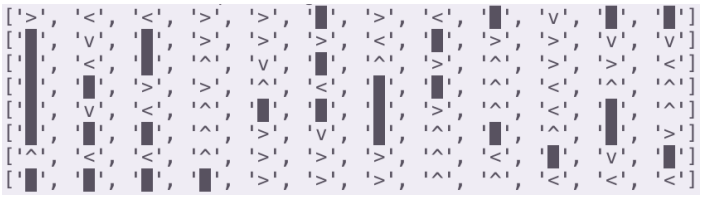


(c) Avec épée et clé

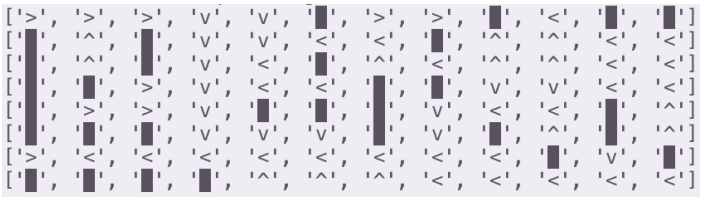


(d) Avec tous les objets

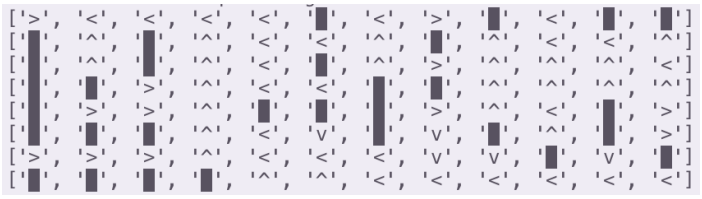
FIGURE 5 – Politique apprise pour le premier niveau.



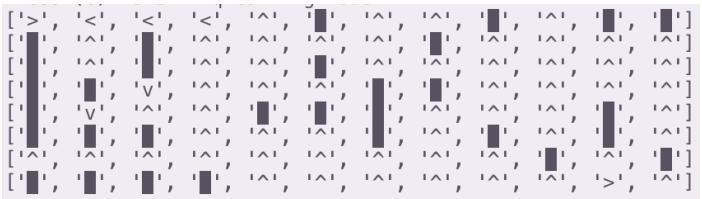
(a) Sans objets



(b) Avec épée



(c) Avec épée et clé



(d) Avec tous les objets

FIGURE 6 – Politique apprise pour le quatrième niveau.