Université de Technologie de Compiègne

UV IA01

Compte-rendu de TP n°2

Noé Connan

Julien Pontoire

Semestre : A23

Table des matières

Introduction

Pour ce TP, nous avons choisi d'utiliser la même représentation que dans le médian. Voici un rappel de la représentation :

```
(setq map '((1 12 2)(2 1 3)(3 2 4)(4 3 5)(5 4 8 6)(6 5 7)
(7 8 6)(8 7 5)(12 13 1) (13 24 12)(15 22)(20 21 29)(21 22 20)
(22 27 21 15)(24 25 13)(25 36 26 24)(26 25 27)(27 26 22)
(29 32 20)(32 29)(36 25)))

(setq horcruxesDescription
        '(("Journal intime de Tom Jedusor"
            (methodeDestruction "Crochet de Basilic"))
        ("Medaillon de Salazar Serpentard"
            (methodeDestruction "Epee de Gryffondor"))
        ("Bague de Gaunt"
            (methodeDestruction "Epee de Gryffondor"))
        ("Coupe de Helga Poufsouffle"
            (methodeDestruction "Crochet de Basilic"))
        ("Nagini"
            (methodeDestruction "Epee de Gryffondor"))
        ("Diademe de Rowena Serdaigle"
            (methodeDestruction "Feudeymon"))))

(setq horcruxesMap '((8 "Journal intime de Tom Jedusor")
                     (12 "Medaillon de Salazar Serpentard")
                     (15 "Bague de Gaunt")
                     (22 "Coupe de Helga Poufsouffle")
                     (26 "Nagini")
                     (29 "Diademe de Rowena Serdaigle")))

(setq armesMap '((3 "Crochet de Basilic")
                 (32 "Feudeymon")
                 (25 "Epee de Gryffondor")
                 (20 "Sortilege de la Mort")))
```

1 Analyse

Harry Potter circule dans le labyrinthe en suivant un parcours en profondeur ; il explore d'abord tous les fils possibles dans une direction (ordre de choix : haut > droit > bas > gauche) puis, s'il a atteint une impasse ou la profondeur maximale de 7, il part explorer la branche suivante, jusqu'à ce que toutes les cases soient explorées.

Lors de notre réflexion sur le parcours en profondeur, nous nous sommes heurtés à un premier problème. Lors du parcours en partant de la case 1, lorsque Harry atteint la case 5 il y a 2 successeurs possibles : 8 et 6. Harry va partir sur la case 8 puis 7 et 6 et ensuite lorsqu'il remonte jusqu'à la case 5 il doit aller dans la case 6. Or cette case a déjà été parcourue dans la branche précédente. Nous avons fait le choix de la parcourir tout de même car si on imagine par exemple que 6 avait eu un successeur en plus, ce successeur n'aurait pas été parcouru par la première branche car à 6 Harry aurait atteint la profondeur maximale mais dans la deuxième branche Harry l'aurait exploré puisqu'il n'aurait pas encore atteint la profondeur maximale.

Voici donc notre arbre de parcours final en partant des cases 1 et 36 :

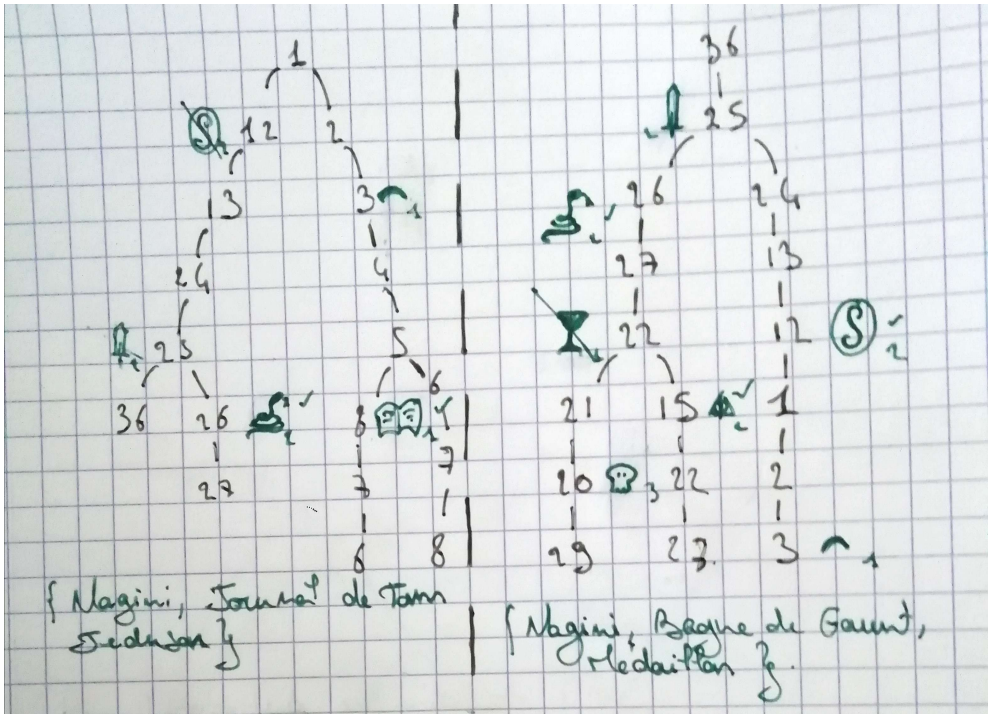

FIGURE 1 – Arbres de parcours de Harry avec Horcruxes et méthodes de destruction

Harry détruit 2 Horcruxes en partant de la case 1 et 3 à partir de la case 36 avec une profondeur maximale de 7. Indépendamment de la profondeur, Harry n'arrivera pas à détruire tous les Horcruxes à partir de ces positions : pour la case 1, il rencontre le Médaillon de Serpentard avant de trouver la méthode adéquate pour le détruire (en 25) et en partant de la case 36 il trouve la Coupe de Helga Poufsouffle avant le Crochet du Basilic.

2 Fonctions de service

Voici les fonctions de services utilisées dans le programme. Au nombre de 6, nous avons décidé d'en ajouter une dédiée à la gestion du déplacement de Voldemort. `successeur-voldemort` prend en successeur potentiel n'importe quelle case blanche du tableau hormis celle sur laquelle Voldemort est déjà, ce qui correspond à son mode de déplacement selon le TD. Les autres fonctinos de services servent respectivement à :

- choisir les successeurs valides pour Harry, afin qu'il ne retourne pas sur ses pas avant d'avoir atteint la profondeur 7 ;
- vérifier si, en passant sur un Horcruxe, les personnages possèdent la méthode pour le détruire ou non (`methode-destruction + has-bonne-arme`) ;
- revenir en arrière dans les codes sans variable globale (`backtracking`) ;
- de vérifier que tous les horcruxes ont été détruits (`allHorcDetruits`)

```
(defun successeurs-valides (case carte chemin)
  (if (and (and (numberp case) (listp carte)) (listp chemin))
      (let ((succ (cdr (assoc case carte))))
        (remove-if #'(lambda (x) (member x chemin)) succ))
    "Erreur de type dans la saisie des parametres") )

(defun methodeDestruction (horc horcDesc)
  (if (and (stringp horc)(listp horcDesc))
      (let ((desc (cadr (assoc horc horcDesc :test #'string=))))
        (if desc
            (cadr desc)) )
    "Erreur de type dans la saisie des parametres"))


(defun hasBonneArme (horc listeM horcDesc)
  (if (and (and (stringp horc)(listp listeM))(listp horcDesc))
      (let ((methode (methodeDestruction horc horcDesc)))
        (if (member methode listeM :test #'string=)
            T))
    "Erreur de type dans la saisie des parametres" ) )

(defun successeurs-voldemort (case carte)
  (if (and (numberp case) (listp carte))
      (let ((succ (mapcar #'car carte)))
        (remove-if (lambda (x) (eql case x)) succ))
```

```
    "Erreur de type dans la saisie des parametres" ))

(defun backtracking (chemin cheminGeneral map)
  (if (and (and (listp chemin) (listp cheminGeneral)) (listp map))
      (let ((chem (reverse chemin)))
        (pop chem)
        (loop while (and (not (successeurs-valides (car chem) map cheminGen
              (pop chem)
              )
        (if (not chem)
            nil
          (reverse chem)))
    "Erreur de type dans la saisie des parametres"
    )
  )

(defun allHorcDetruits (listH listHrecup)
  (if (and (listp listH) (listp listHrecup))
      (let ((listeHorc (mapcar #'cadr listH)) (test 0))
        (loop while (and (eq 0 test) (> (length listeHorc) 0)) do
              (if (member (car listeHorc) listHrecup :test #'string=)
                  (pop listeHorc)
                (setq test 1))
              )
        (if (eq test 0)
            T
          nil)
        )))
```

3 Recherche en profondeur pour la recherche de Harry Potter

Nous n'avons pas réussi à produire un code entièrement fonctionnel pour la recherche en profondeur de Harry sans faire usage de variable globale, qui est une mauvaise pratique de code : nous vous proposons donc deux codes différents pour cette partie. Dans un premier temps un code utilisant des variables globales et qui s'exécute entièrement et traite toute la recherche et dans un second temps un code sans variable globale mais qui ne traite pas entièrement la recherche à cause d'un problème d'algorithme.

3.1 Fonction utilisant des variables globales

3.1.1 Algorithme

```
recherche_en_profondeur_Harry:

Initialisation des variables globales.
Initialisation des variables avec caseDepart.

    Si tous les horcruxes n'ont pas déjà été détruits
        Afficher "Harry Potter est actuellement sur caseDepart"
        Si caseDepart n'est pas déjà dans chemin, l'ajouter;
        Si armeSurCase n'est pas élément de armesCollectees :
            l'ajouter dans armesCollectees;
            Afficher "Harry Potter a trouvé armeSurCase"

        Si horcruxeSurCase n'est pas dans HorcruxeDetruit ni dans
        horcruxesIncassables :
            Si armesCollectees contient la description adéquate :
                L'ajouter dans HorcruxeDetruit;
            Sinon :
                L'ajouter dans HorcruxeIncassable;
        Si length de chemin >= 8 :
            Afficher "Fin de la recherche de cette branche";
            Afficher le chemin parcouru;
            successeur();
            Afficher "Harry se déplace en successeur";

        recherche_en_profondeur_Harry(successeur);
```

3.1.2 Code LISP

```
(defvar *horcruxesDetruits* '())
(defvar *armesCollectees* '())
(defvar *horcruxesIncassables* nil)
```

```
(defun code-Harry (case)
; permet de faire la recherche et l'affichage en une seule instruction
  (setq *horcruxesDetruits* '()) ; permet de vider les variables globales
  (setq *armesCollectees* '())
  (setq *horcruxesIncassables* '())
  (recherche-en-profondeur-H case map horcruxesDescription armesMap
  horcruxesMap '())
  (format t "~%Resultat final~%")
  (format t "~%Horcruxe(s) Detruit(s) : ~s ~%Methode(s) Trouvee(s) : ~s"
  *horcruxesDetruits* *armesCollectees*)
  "Fin"
  )
```

Pour cette partie, nous utilisons donc 3 variables globales qui permettent de définir la liste des armes possédées, les horcruxes détruits et la liste des horcruxes plus atteignables par Harry. Cette partie se décompose en 2 fonctions, la première va réaliser la recherche en profondeur et la seconde va appeler la première fonction et réaliser un affichage. Nous n'avons pas mis le code de la recherche en profondeur sur ce rapport mais vous pouvez le retrouver sur notre fichier TP2.cl.

Nous étions originellement partis sur l'idée de faire un algorithme récursif mais nous nous sommes heurtés à différents problèmes qui font que nous avons décidé d'utiliser des variables globales. Le premier de ces problèmes est le fait de garder en mémoire les armes récupérées et les horcruxes détruits dans une branche. En effet avec un algorithme récursif classique, lorsque le parcours d'une branche est terminé, toutes les variables qui ont été mises à jour à l'intérieur de cette branche sont réinitialisées à leurs valeurs par défaut à l'entrée de la branche. Une solution aurait été de mettre les données souhaitées en retour de la fonction mais nous nous heurtions à d'autres problèmes. Nous avons choisi de faire une autre fonction à côté pour gérer l'affichage car la récursivité ferait que l'affichage se répéterait plusieurs fois au cours de l'exécution.

3.2 Fonction sans variable globale

3.2.1 Algorithme

```
Algorithme_Harry:

Initialisation des variables.
Ajout de la caseDepart à la liste des successeurs

    Tant qu'il y a des successeurs et profondeur > 0 et tous les horcruxes*
```

```
    n'ont pas été détruits
        Ajouter (car successeurs) à chemin et à cheminGeneral
        case <- (pop successeurs)
        Afficher "Harry Potter est actuellement sur case"
        prof <- prof + 1
        Si armeSurCase et elle n'est pas élément de armesCollectees :
            l'ajouter dans armesCollectees;
            Afficher "Harry Potter a trouvé armeSurCase"

        Si horcruxeSurCase n'est pas dans HorcruxeDetruit et que la case
        n'a pas déjà été visitée:
            Si armesCollectees contient la description adéquate :
                L'ajouter dans HorcruxeDetruit;
            Sinon :
                L'ajouter dans HorcruxeIncassable;

        succ <- les successeurs valides de case

        Si prof < la profondeur max et succ :
            On ajoute les éléments de succ dans successeurs
        Sinon
            Afficher : "Parcours de branche terminé"
            chemin <- chemin sans la dernière branche
            prof <- longueur de chemin - 1

renvoie armesCollectees et horcDetruits;
```

3.2.2 Code LISP

Pour cette version, nous sommes partis sur une fonction non pas récursive mais itérative à l'aide d'une boucle while. Cette fonction itère sur la première case puis sur ses successeurs et réalise les mêmes tests que la fonction avec variables globales. Ici nous nous heurtons cependant à un problème : lorsqu'on est à la fin d'une branche, il faut trouver un moyen de revenir au bon endroit dans la branche pour enchaîner sur la suivante. Pour cela nous avons réalisé une fonction de backtracking qui permet de revenir au bon endroit dans la branche. Cette fonction ne traite que le chemin, il aurait été possible de lui faire traiter également la profondeur mais nous avons choisi de le faire nous même dans la fonction de recherche en réalisant une soustraction entre les longueurs des chemins. Grâce à la

réalisation de la recherche à l'aide d'une boucle while, le problème posé par la récursivité est réglé. Cependant nous nous sommes heurtés à un autre problème que nous n'avons pas réussi à résoudre. Lorsque l'on prend pour case de départ 1 la fonction va comme prévu réaliser la recherche sur la partie droite du labyrinthe sans problème puis va passer dans la branche 1-2-3-4-5-8-7-6. Cependant en sortant de cette dernière la fonction devrait en arrivant à la case 5 aller dans la case 6. Or il ne le fait pas car selon lui 6 n'est pas un successeur valide puisqu'il a déjà été visité. Nous avons essayé différentes méthodes pour résoudre ce problème mais aucune n'a fonctionné ou alors elle amenait des problèmes à d'autres niveaux de la fonction tels qu'une boucle infinie.

4 Lord Voldemort part à la recherche des Horcruxes

Nous avons pour cette partie également deux codes différents, un réalisé avec des variables globales et un sans en utiliser. Pour le code sans variables globales, nous sommes confrontés au même problème de non traitement de la dernière branche.

4.1 Fonction avec des variables globales

4.1.1 Algorithme

```
algorithme_Voldemort:

Initialisation des variables globales.

    Si tous les horcruxes n'ont pas déjà été détruits et si Harry
    n'a pas été tué
        Afficher "Harry Potter est actuellement sur caseDepart"
        définition des variables
        Si caseH n'est pas déjà dans cheminH, l'ajouter;
        Si armeSurCaseH n'est pas élément de armesCollecteesH
        ni de armesCollecteesV :
            l'ajouter dans armesCollecteesH;
            Afficher "Harry Potter a trouvé armeSurCaseH"

        Si horcruxeSurCaseH n'est pas dans HorcruxeDetruitH ni dans
        horcruxesIncassables ni dans HorcruxeDetruitV :
            Si armesCollecteesH contient la description adéquate :
                L'ajouter dans HorcruxeDetruitH;
            Sinon :
```

```
            L'ajouter dans HorcruxeIncassable;

        Si la longueur du cheminV est inférieure à celle de cheminH
            on ajoute caseV à cheminV
            Si armeSurCaseV n'est pas élément de armesCollecteesV
            ni de armesCollecteesH :
                l'ajouter dans armesCollecteesV;
                Afficher "Voldemort a trouvé armeSurCaseH"

            Si horcruxeSurCaseV n'est pas dans HorcruxeDetruitV ni dans
            HorcruxeDetruitH et que caseV n'est pas dans (butlast cheminV):
                Si armesCollecteesV contient la description adéquate :
                    L'ajouter dans HorcruxeDetruitV;

            Afficher la liste des cases accessibles pour Voldemort
            Demander à l'utilisateur la case

            Si Voldemort possède le sortilège de mort et qu'il est sur la
            même case que Harry :
                L'ajouter dans horcruxeDetruitV
                ajouter la nouvelle case de Voldemort à son chemin

        Si length de chemin >= 8 :
            Afficher "Fin de la recherche de cette branche";
            Afficher le chemin parcouru;
            successeur();
            Afficher "Harry se déplace en successeur";

        Algorithme_Voldemort;
```

4.1.2 Code LISP

```
(defvar *horcruxesDetruitsH* '())
(defvar *armesCollecteesH* '())
(defvar *horcruxesIncassables* '())
(defvar *horcruxesDetruitsV* '())
(defvar *armesCollecteesV* '())
```

```
(defvar *cheminV* '())
(defvar *result* nil)

(defun code-voldemort-glob (caseH caseV)
  (setq *cheminV* '())
  (setq *result* caseV)
  (setq *horcruxesDetruitsH* '()) ; permet de vider les variables globales
  (setq *armesCollecteesH* '())
  (setq *horcruxesIncassables* '())
  (setq *horcruxesDetruitsV* '())
  (setq *armesCollecteesV* '())
  (voldemort-glob caseH caseV map horcruxesDescription armesMap
  horcruxesMap '())
  (format t "~%Resultat final~%")
  (format t "~%Harry : Horcruxe(s) Detruit(s) : ~s
~~%Methode(s) Trouvee(s) : ~s" *horcruxesDetruitsH* *armesCollecteesH*)
  (format t "~%Voldemort : Horcruxe(s) Detruit(s) : ~s
~~%Methode(s) Trouvee(s) : ~s" *horcruxesDetruitsV* *armesCollecteesV*)
  (format t "~%Chemin de Voldemort : ~s" *cheminV*)
  "FIN"
  )
```

Pour réaliser cette partie du TP, nous étions originellement également partis sur une réalisation avec des variables globales. En effet nous nous hertions aux même problèmes qu'avec la partie précédente pour la recherche en profondeur de Harry mais d'autres problèmes s'ajoutaient également comme le fait de gérer le chemin de Voldemort lors des appels récursifs. Lorsqu'une branche de Harry se serait terminée, le chemin de Voldemort serait revenu en arrière, ce qui n'est pas souhaitable. Nous avons donc des variables globales qui permettent de stocker les horcruxes détruits et les armes possédées pour chacun des deux personnages, la liste des horcruxes incassables pour Harry, le chemin de Voldemort et la case suivante de Voldemort qui est entrée par l'utilisateur. Dans cette partie, il est important de vérifier aux bons moments si Voldemort a tué Harry. En effet étant donné qu'il ne se déplace quand quand la profondeur de Harry augmente, il est important de faire 2 vérifications : une générale et une autre quand Voldemort ne s'est pas déplacé.

4.2 Fonction sans variables globales

4.2.1 Algorithme

`Algorithme_Harry:`

```
Initialisation des variables.
Ajout de la caseDepart à la liste des successeurs

    Tant qu'il y a des successeurs et profondeur > 0 et tous les horcruxes
    n'ont pas été détruits et Harry n'est pas mort
        Ajouter (car successeurs) à cheminH et à cheminGeneralH
        caseH <- (pop successeurs)
        Afficher "Harry Potter est actuellement sur caseH"
        profH <- profH + 1
        Si prof-max de Harry < profH
            prof-max de Harry += 1

        Si armeSurCaseH n'est pas élément de armesCollecteesH ni de
        armesCollecteesV:
            l'ajouter dans armesCollecteesH;
            Afficher "Harry Potter a trouvé armeSurCaseH"

        Si horcruxeSurCaseH n'est pas dans HorcruxeDetruitH ni dans
        HorcruxeDetruitV et que la case n'a pas déjà été visitée:
            Si armesCollecteesH contient la description adéquate :
                L'ajouter dans HorcruxeDetruitH;
            Sinon :
                L'ajouter dans HorcruxeIncassable;

        Si profV < prof-max de Harry :
            ajouter caseV dans cheminV
            afficher "Voldemort se trouve sur la caseV"

            Si armeSurCaseV n'est pas élément de armesCollecteesV
            ni de armesCollecteesH :
                l'ajouter dans armesCollecteesV;
                Afficher "Voldemort a trouvé armeSurCaseH"

            Si horcruxeSurCaseV n'est pas dans HorcruxeDetruitV ni dans
            HorcruxeDetruitH et que caseV n'est pas dans (butlast cheminV):
                Si armesCollecteesV contient la description adéquate :
                    L'ajouter dans HorcruxeDetruitV;
```

```
            Afficher la liste des cases accessibles pour Voldemort
            Demander à l'utilisateur la case

            Si Voldemort possède le sortilège de mort et qu'il est sur la
            même case que Harry :
                L'ajouter dans horcruxeDetruitV
                ajouter la nouvelle case de Voldemort à son chemin


        succ <- les successeurs valides de case

        Si prof < la profondeur max et succ :
            On ajoute les éléments de succ dans successeurs
        Sinon
            Afficher : "Parcours de branche terminé"
            chemin <- chemin sans la dernière branche
            prof <- longueur de chemin - 1

renvoie armesCollecteesH, horcDetruitsH, armesCollecteesV et horcDetruitsV;
```

4.2.2 Code LISP

Pour cette partie, nous avons repris la même base que le code pour la recherche en profondeur de Harry sans variables globales, en dédoublant les variables que nous avions initialisé dans le let pour en avoir pour Harry et pour Voldemort. La fonction se base énormément sur la fonction de recherche en profondeur de harry sans variables globales et la fonction de Voldemort avec variables globales.

5 Créativité

Par manque de temps, nous n'avons pas implémenté nos idées en LISP. La partie ci-dessous sera donc essentiellement des réflexions de mécaniques à ajouter agrémentées d'un algorithme succin lorsque c'est pertinent.

5.1 Amélioration des règles

Pour assurer une forme de cohérence par rapport à la saga Harry Potter, il nous paraît à la fois simple et logique que Voldemort ne détruise pas les Horcruxes autres que Harry Pot-

ter. Pour ce faire, et comme la mort de Harry Potter n'est pas gérée de la même façon que les Horcruxes, il suffit de créer une fonction alternative à `as-bonne-arme| afin que, lorsqu'il renco` de destruction pour empêcher Harry de s'en servir.

Une autre amélioration serait que pour réellement gagner, après avoir détruit tous les Horcruxes sauf lui-même, Harry Potter doive trouver la case de Voldemort. Cette condition prévaudrait sur la condition de victoire de Voldemort avec le sort de mort.

5.2 Personnages

5.2.1 Dobby

Dans Les Chroniques de la Mort, Livre 1, Dobby l'elfe de maison meurt en se sacrifiant pour sauver Harry. Nous trouvions qu'il était amusant d'ajouter un facteur aléatoire qui rende le labyrinthe un peu plus facile, incarné par Dobby. Il se déplace de la même manière que Harry sans pour autant détruire les Hurcruxes ni récupérer les méthodes ; et si Voldemort tente de tuer Harry et que Dobby est à 2 cases ou moins (en distance de Manhattan) de lui, Doby meurt à la place de Harry et le parcours en profondeur de Harry continue normalement.

Pour implémenter Dobby, il faudrait idéalement créer une fonction de service de calcul de distance ; le problème est que la disposition des cases ne permet pas de facilement calculer la distance, il faut ajouter une mapDistance. Je propose le formalisme suivant : (caseA caseB distance) ;

```
(setq mapDistance ' (1 1 0) (1 2 1) (1 3 2) ...
(1 12 1) ... (2 11 1 ) ...)

Distance (caseA caseB)
Cherche la case contenant caseA, caseB en 1e et 2e position;
Retourne son dernier élément;

Dobby :
    Initialisation (caseD)
    Tant que "Harry Potter" n'est pas dans horcruxesDetruits :
        Dobby (successeur)
    Si Distance (caseD, caseV) = 0 :
        Afficher "Dobby est mort";
        Fin.
    Si Distance (caseH, caseV) = 0 :
        Si "Sortilège de la Mort" est dans armesCollecteesV :
```

```
Si Distance (caseD, caseH = 0 ou 1 ou 2) :
    Afficher "Dobby s'est sacrifié pour sauver Harry";
    Retirer Harry de la liste horcruxesDetruits;
    Fin.
```

La mise en place du sauvetage de Harry sous ce mode est un peu délicat car il doit intervenir avant la prochaine évaluation de la liste horcruxesDetruits. Il faudrait faire attention à l'ordre des opérations entre Voldemort et Dobby dans l'implémentation.

5.2.2 Dumbledore

De façon générale, Albus Dumbledore est l'un des personnages les plus chaotiques de la série : il cache tout le temps des choses à Harry, il fait prendre à ses élèves mineurs des risques complètement irresponsables, etc. Il est aussi au courant de l'existence des Horcruxes. Dumbledore se déplace exactement comme Voldemort (sur n'importe quelle case à chaque fois que le parcours de Harry augmente de 1 en profondeur) et, si il tombe sur un Horcruxe, il le déplace à un endroit aléatoire qui n'est pas une case grise.

6 Conclusion

6.1 Analyse critique

La principale faiblesse de la représentation utilisée est qu'elle ne permet pas un calcul facile de la distance de Manhattan ; il faudrait que les cases soient indexées avec une ordonnée et une abscisse, ce qui permettrait le calcul de distance et donc :

- d'améliorer le parcours du labyrinthe - via un algorithme glouton ou A* par exemple. Ainsi Harry pourrait s'orienter directement vers les Horcruxes et être un peu plus "intelligent" ;
- de faciliter l'impémentation de la rencontre Harry-Voldemort ;
- de rendre possible une implémentation propre de Dobby.

Notre code garde un certain nombre de défauts dont la présence de variables globales ; certaines difficultés sont aussi dues au langage (ou au moins à une méconnaissance du langage) car la partie créativité serait grandement facilitée si on pouvait faire de la Programmation Objet en LISP.

6.2 Miscellanious

À défaut d'y avoir pleinement réfléchi, nous voulions partager d'autres idées saugrenues à implémenter en LISP :

- Rogue est un agent double acariâtre, qui a toujours l'air passablement irrité, et n'aime pas trop Harry. Il serait parfait en "case grise" mobile si le labyrinthe n'en

comportait pas déjà autant. Il se déplacerait sur des cases adjacentes à chaque fois que Harry avance dans l'arbre (donc plus souvent que les autres personnages).

— Nous devrions donc avoir plusieurs labyrinthes possibles qui soient ou plus grands, ou moins contraignants (moins de cases grises), et augmenter la profondeur en conséquence afin d'augmenter la rejouabilité. Pour ce faire, il suffit de changer la liste initiale `map`.