IA01

# TP02 : Harry Potter et les reliques de la mort

Alexis Deseure et Colin Manyri

# Table des matières

# I. Introduction

## Organisation du rapport

Ce rapport a pour objectif de présenter ainsi que d'argumenter le code LISP associé au TP02. Vous trouverez le détail du code source ainsi qu'une présentation pour chaque question du TP. Ce document s'inscrit dans l'archive TP2_Deseure_Manyri.zip où se trouve le fichier source de l'exercice contenant les fonctions ainsi que quelques exemples et des jeux de testes.

Pour chaque question, une description de son objectif ainsi que de la manière dont elle a été traitée sera présentée et un exemple de son fonctionnement sera abordé dans ce rapport.

## Présentation du TP02

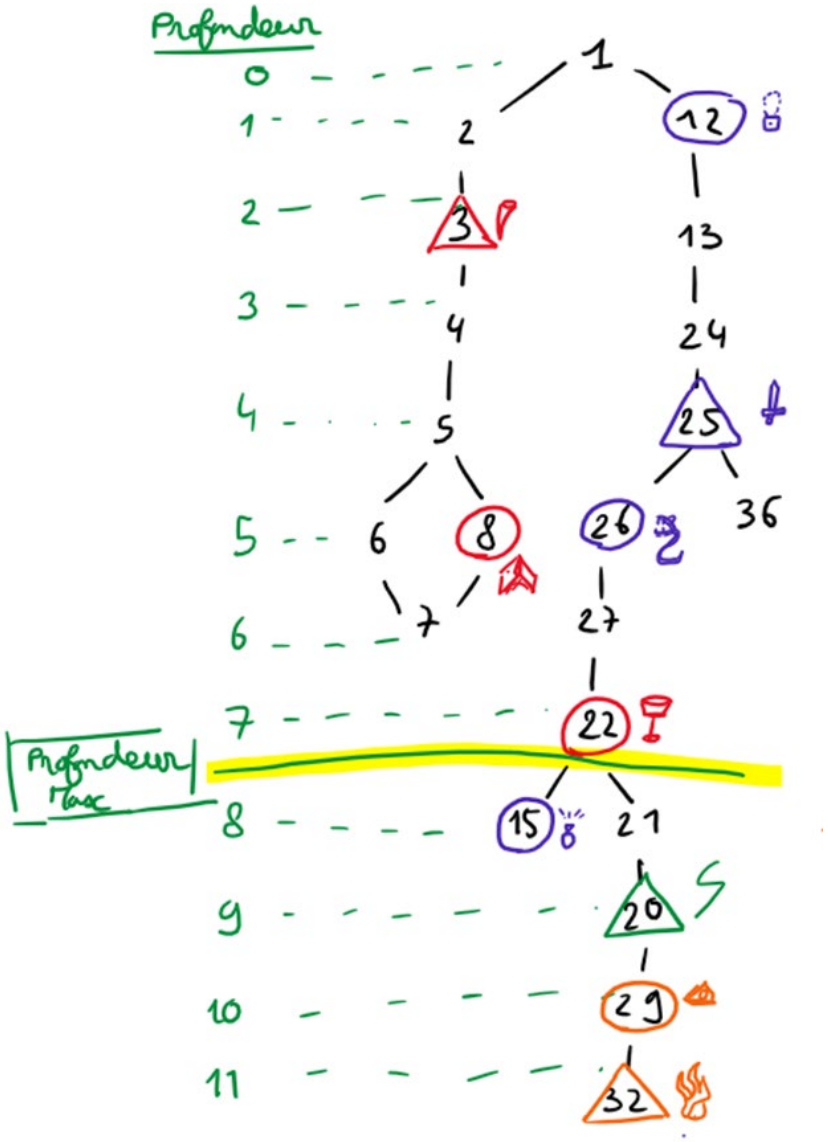
Ce TP aborde le thème de la recherche dans un espace d'états grâce à une mise en situation dans l'univers de Harry Potter, une série de livre écrits par l'auteure britannique J.K. Rowling.
Le sujet met en scène une chasse aux Horcruxes, des objets maléfiques, que le personnage principale, Harry Potter, cherche à détruire.
Durant tout le sujet, Harry se déplace dans un espace d'état comportant les Horcruxes et les moyens de les détruire. Par la suite, le sujet rajoute le personnage de Voldemort, le principal antagoniste de la série, qui lui aussi cherchera à récupérer les Horcruxes (ils correspondent en fait à des vies supplémentaires).
Enfin, le sujet introduit une partie « créativité », qui nous invite à complexifier encore le modèle en effectuant, par exemple, des ajouts en liens avec le thème de la saga.

# II. Analyse

## Question 1 et 2

Les armes sont représentées par des triangles et les Hocruxes par des cercles. Chaque Horcurxe est de la même couleur que l'arme qui permet de le détruire. Des pictogrammes ont été rajouté et font référence au nom de chaque arme et Horcurxes. Enfin, les nombres en verts indique la profondeur de l'arbre, sachant que l'exploration s'arrête à la profondeur 7 et que le point de départ a une profondeur de 0.

Pour le départ de 1 :




Le chemin du parcoure en profondeur avec profondeur maximum de 7 pour ce premier arbre est :

(1 2 3 4 5 6 7 8 12 13 24 25 26 27 22 36)

On récupère dans l'ordre les Armes et Horcuxes suivants :

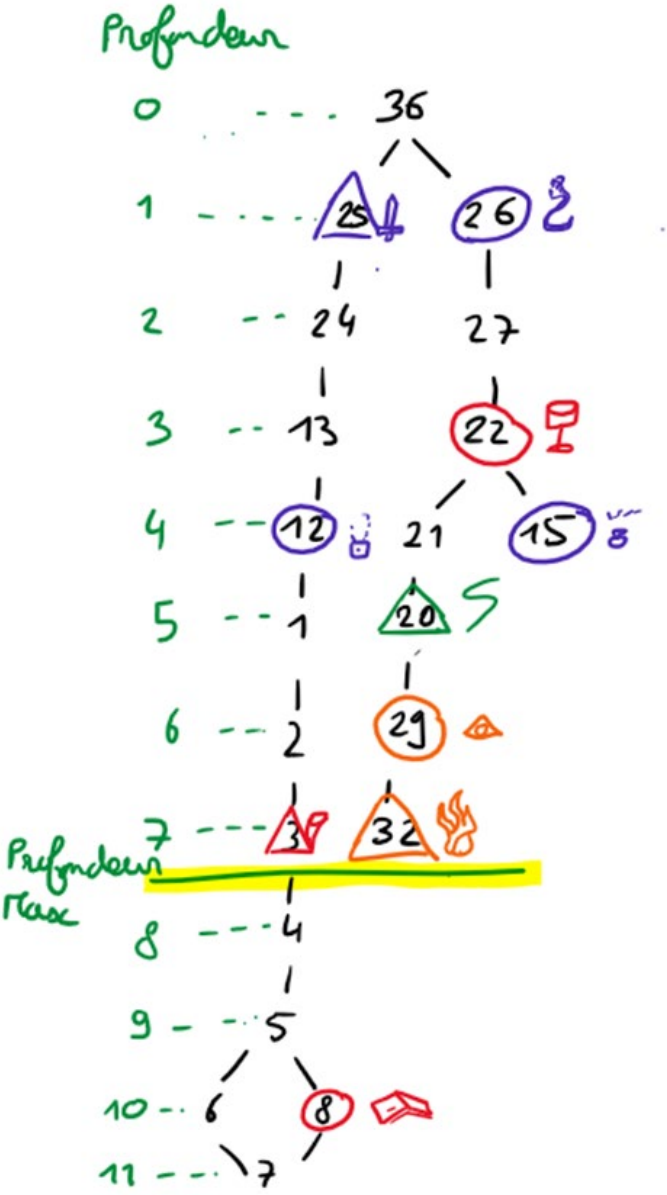
3° Crochet de basilic

8° Journal de Tom Jedusor -> detruit par Crochet de basilic

25° Epée de Gryffondor

26° Nagini -> detruit par Epée de Gryffondor

Pour le départ de 36



L'ordre de parcourt en profondeur des cases est le suivant :

(36 25 24 13 12 1 2 3 26 27 22 21 20 29 32 15)

Dans l'ordre on récupère les armes et Horcurxes suivants :

25° Epée de Gryffondor

12° Médaillon de Salazar Serpentard -> détruit par Epée de Gryffondor

3° Crochet de basilic

26° Nagini -> détruit par Epée de Gryffondor

22° Coupe de Helga Poufsouffle -> détruit par Crochet de basilic

20° Sortilège de Mort

32° Feudeymon

15° Bague de Gaunt

## Question 3

Dans les deux cas on ne peut pas détruire tous les Horcruxes même sans profondeur limité. Nous sommes toujours obligés de passer par la case 29 où se trouve le diadème de Rowena Serdaigle pour rejoindre la case 32 où se trouve la méthode de destruction du diadème : Feudeymon. On ne peut donc jamais détruire le diadème en partant soit de 1 soit de 36. Plus généralement, dans le cas d'un départ sur 1, on détruit tous les Horcruxes excepté le Médaillon de Salazar Serpentard et le diadème de Rowena Serdaigle et on récupère toutes les armes. Dans le cas d'un départ sur la case 36, toutes les armes sont récupérées et seul le diadème n'est pas détruit.

# III. Fonctions de services

## Choix de la représentation

```
(setq carte '((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")))

(setq reliqueMap '((6 "Baguette de sureau") ;utile pour la partie créativité
  (36 "Cape d'invisibilite")
        (15 "Pierre de resurrection")))
```

## Question 1 : fonction successeurs-valides

Cette fonction renvoie la liste des successeurs valides de point d'après la carte c'est-à-dire les successeurs de points qui ne sont pas dans chemin.

On récupère d'abord la liste de tous les successeurs de point dans une liste listesuccesseurs. Puis pour chaque successeur ainsi récupéré on vérifie qu'il n'est pas dans chemin.

```
;successeurs-valides(int list list) -> list
```

```
(defun successeurs-valides (point carte chemin)
       (if (and (listp carte) (listp chemin)) ;évite une erreur si les arguments ne sont pas
corrects

              (let ((valides) (listesuccesseurs (assoc point carte))) ;initialisation d'une
liste vide et d'une liste de successeurs

                     (reverse (dolist (successeur (cdr listesuccesseurs) valides) ; pour tous
les successeurs du point dans carte
                                   (if (not (member successeur chemin)) ; si on n'est pas déjà passe
par ce point (présence dans chemin)
                                          (push successeur valides) ; on ajoute ce point dans la
liste valides
                                   )
                     )) ;on utilise le reverse pour correspondre à la consigne "il (Harry) ne
pourra se déplacer que sur des cases contiguës (en haut, à droite, en bas ou à gauche)"
              )
              (format t "Erreur dans carte ou dans chemin. Veuillez donner des listes. ~%")
       )
)
```

## Question 2 : fonction methodeDestruction

La fonction methodeDestruction prend en argument le nom d'un Horcruxe et la description des méthodes de destructions de tous les Horcruxes et renvoie la méthode de destruction pour l'Horcruxe donné en argument.

Cette fonction teste d'abord si l'Horcruxe en argument fait partie de la liste des Horcruxes décrits. Puis, si c'est le cas, il va chercher dans la liste la bonne methode de destruction. Dans le cas contraire, la fonction renvoie nil et le message : `"Pas d'information sur l'Horcruxe : nom de L'Hocruxe dans listedescrition`

```
;methodeDestruction( char list) -> char

(defun methodeDestruction (horcruxe listedescriptions)
       (let ((description_horcruxe (assoc horcruxe listedescriptions :test
#'string=))) ;trouver la description du bon Horcruxe avec assoc
              (if description_horcruxe ;si cette description existe
                     (cadr (assoc 'methodeDestruction (cdr description_horcruxe)))
;On fait un autre assoc pour determiner la methode de destruction de l'horcuxe
                     (format t "Pas d'information sur l'Horcruxe : ~a dans ~a !~%"
horcruxe listedescriptions) ;sinon il n'y a pas d'information sur l'horcruxe
              )
       )
)
```

## Question 3 : fonction hasBonneArme

La fonction hasBonneArme teste si les armes possédées par un personnage permettent de détruire l'Horcruxe donné en argument. La fonction renvoie True si c'est le cas et Nil sinon.

La fonction hasBonneArme utilise la fonction methodedestruction pour recupérer la méthode de destruction adapté à l'Horcruxe. Puis verifie que cette méthode est dans la liste "armepossedees".

```
;hasBonneArme (char list list(list)) -> bool
;exemple : (hasBonneArme "Nagini" '("Epee de Gryffondor") horcruxesDescription)
renvoie True

(defun hasBonneArme (horcruxe armespossedees description)
      (if (and (listp armespossedees) (member (methodeDestruction horcruxe
description) armespossedees :test #'string=))
            T ;teste si armespossedees est bien une liste et si la methode de
            destruction necessaire est presente dans armespossedees
      )
)
```

# IV. Recherche en profondeur pour Harry Potter

Pour cette partie, nous avons choisi de ne pas créer de fonctions intermédiaires (autres que les fonctions de services définis précédemment) pour répondre pleinement au sujet qui sous-entend qu'il y a une seule fonction, mais également car c'est ainsi qu'on l'a initialement créée. Dans la partie créativité, qui rajoute plus de fonctionnalités, on retrouvera des fonctions intermédiaires pour chacune d'elles.

## Algorithme et explications

Harry Potter étant lui-même un Horcruxe, il lui est impossible de le savoir et de penser à se détruire lui-même. Le seul moyen pour lequel le programme se termine est lorsqu'il a parcouru la carte selon une profondeur maximale (ici 7) et qu'il soit revenu à sa position initiale.

L'algorithme en langage naturel ne contient pas les affichages consoles (type print ou format en Lisp). Pour expliquer certaines lignes de l'algorithme, on écrira entre parenthèses les informations importantes :

On définit la fonction exploreprofondeur :

- Paramètres de la fonction :

carte -> contient la carte suivant le format de l'énoncé et respectant l'ordre (haut, droite, bas et gauche)
horcruxesDescription -> contient le nom et la description des Horcruxes (ie : l'arme nécessaire pour les détruire, cf. énoncé)
horcruxesMap -> contient la liste des emplacements des Horcruxes (cf. énoncé)
armesMap -> contient la liste des emplacements des armes (cf. énoncé)
profondeurMax -> entier (>0) correspondant à la profondeur max (mettre 7 dans notre cas)
pointdepart -> entier correspondant au point de départ d'Harry (si ce n'est pas une case correcte le programme se stoppera tout de suite avec seulement le chemin contenant la mauvaise case)
chemin (optionel) -> ce paramètre contient la liste des positions déjà parcourues par Harry et ne dois pas être remplie initialement pour avoir le fonctionnement attendu
horcruxedetruit (optionel) -> de la même manière il ne doit pas être fourni initialement et contiendra la liste des Horcruxes détruits
armespossedees (optionel) -> identique à horcruxedetruit mais pour les armes

- Algorithme :
Définir :

- arme comme la chaine de caractères contenant le nom de la potentielle arme présente sur pointdepart
- horcruxe comme la chaine de caractères contenant le nom du potentiel Horcruxe présent sur pointdepart
- brancheActuelle à NIL (il contiendra les valeurs retournés par les appels récursifs de la fonction
- successeurs comme la liste résultante de l'appel de la fonction successeurs-valides avec les arguments pointdepart, carte et chemin

Ajouter pointdepart à la fin de la liste chemin
Si arme n'est pas NIL alors
    Ajouter arme dans armespossedees
Fin SI
(Ce premier bloc permet la détection d'une potentielle arme sur la case actuelle)
Si horcruxe n'est pas NIL alors
    Si le résultat de la fonction hasBonneArme avec les paramètres horcruxe, armespossedees et horcruxesDescription n'est pas NIL alors
        Ajouter horcruxe dans horcruxedetruit
    Fin Si
Fin SI
(Ce bloc permet la détection d'un potentiel Horcruxe sur la case actuelle)
Si profondeurMax != 0 alors
    Si successeurs n'est pas NIL
        Pour chaque position succ de successeurs
            Si succ appartient au resultat de l'appel de successeurs-valides avec les paramètres pointdepart, carte et chemin (cette condition est nécessaire dans le cas d'un croisement des branches par la suite pour éviter de retourner sur des case déjà parcourues)
                Attribuer à brancheActuelle le resultat de l'appel de la fonction actuelle exploreprofondeur avec les paramètres carte, horcruxesDescription, horcruxesMap, armesMap, profondeurMax – 1, succ, chemin, horcruxedetruit et armespossedees
                Modifier horcruxedetruit avec la valeur correspondante dans brancheActuelle
                Modifier armespossedees avec la valeur correspondante dans brancheActuelle
                Modifier chemin avec la valeur correspondante dans brancheActuelle et ajouter la valeur de pointdepart (pour que le chemin contienne le vrai parcourt sans « sauts »)
            Fin Si
        Fin Pour
    Fin Si
Fin Si

(Ce bloc permet le déplacement d'Harry et de parcourir toutes les branches si il y a plusieurs fils au potentiel père actuel (décrémentation de 1 en profondeur). Puis elle permet de modifier les variables qui étaient les paramètres optionnels pour les retourner avec les bonnes valeurs ajoutées par le parcourt)
**Retourner la liste contenant horcruxedetruit, armespossedees et chemin** (pour l'implémentation on utilisera des clés pour pouvoir y accéder plus facilement).

## Code Lisp

Le code Lisp associé à l'algorithme précédent étant relativement long, il est disponible dans le fichier avec l'extension .lisp de l'archive.

# V. Lord Voldemort à la recherche des Horcruxes

## Algorithme et explications

Le principe de cet algorithme est légèrement différent au précédent. En effet, cette fois, un nouveau personnage est introduit : Voldemort, qui rajoute une fin alternative au premier programme. Harry peut toujours finir en ayant parcouru toute la carte, car on le rappel il n'aura pas détruit tous les Horcruxes : il reste toujours lui-même. Sinon, il peut aussi tomber sur Voldemort pendant son parcourt (dont la position est demandée à l'utilisateur à chaque tour parmi les cases adjacente à la sienne). Dans ce cas, si Voldemort possède l'arme du Sortilège de la mort, alors il tue Harry et la partie se termine (même si c'était le dernier Horcruxe, dans tous les cas il resterait encore la vie « principale » de Voldemort donc sur ce modèle Voldemort ne meurt jamais). Ce modèle autorise aussi Voldemort à détruire ses propres Horcruxes et à ramasser des armes selon l'énoncé. (Pour un soucis de cohérence dans la partie créativité, Voldemort ne détruira plus les Horcruxes mais les cachera avec lui pour éviter que Harry ne les détruise (il n'aura pas besoin d'armes pour le faire). Il ne pourra donc pas ramasser d'armes mis à part le sortilège de la mort).

L'algorithme en langage naturel ne contient pas les affichages consoles (type print ou format en Lisp). Pour expliquer certaines lignes de l'algorithme, on écrira entre parenthèses les informations importantes :

On définit la fonction exploreprofondeurVoldemort:

- Paramètres de la fonction :

carte -> contient la carte suivant le format de l'énoncé et respectant l'ordre (haut, droite, bas et gauche)
horcruxesDescription -> contient le nom et la description des Horcruxes (ie : l'arme nécessaire pour les détruire, cf. énoncé)
horcruxesMap -> contient la liste des emplacements des Horcruxes (cf. énoncé)
armesMap -> contient la liste des emplacements des armes (cf. énoncé)
profondeurMax -> entier (>0) correspondant à la profondeur max (mettre 7 dans notre cas)
pointdepartHarry et pointdepartVdm -> entiers correspondant au point de départ d'Harry et de Voldemort (si ce n'est pas une case correcte pour Harry le programme se stoppera tout de suite (après avoir choisi où va Voldemort) avec seulement le chemin contenant la mauvaise case. Si la case n'est pas correcte pour Voldemort, le programme demandera à l'utilisateur de choisir parmi NIL, ce qui renverra une erreur : c'est à l'utilisateur ou future interface d'enter les paramètres correctes pour avoir le résultat attendu)
cheminHarry et cheminVdm(optionel) -> ces paramètres contiennent la liste des positions déjà parcourues par Harry et respectivement Voldemort et ne doivent pas être remplie initialement pour avoir le fonctionnement attendu

horcruxedetruitHarry et horcruxedetruitVdm (optionel) -> de la même manière ils ne doivent pas être fournis initialement et contiendront la liste des Horcruxes détruits par chacun des personnages
armespossedeesHarry et armespossedeesVdm (optionel) -> identique à horcruxedetruitHarry et horcruxedetruitVdm mais pour les armes

- Algorithme :
Définir :
    - armeHarry et armeVdm comme les chaines de caractères contenant le nom des potentielles armes présentent sur pointdepartHarry et pointdepartVdm
    - horcruxeHarry et horcruxeVdm comme les chaines de caractères contenant le nom des potentiels Horcruxes présents sur pointdepartHarry et pointdepartVdm
    - brancheActuelle à NIL (il contiendra les valeurs retournés par les appels récursifs de la fonction
    - caseVdm à NIL (qui contiendra la case entrée par l'utilisateur à entrer dans les paramètres de l'appel récursif pour le déplacement de Voldemort)
    - successeursHarry et successeursVdm comme les listes résultantes de l'appel de la fonction successeurs-valides avec les arguments pointdepartHarry, carte et cheminHarry pour successeursHarry et pointdepartVdm, carte et NIL pour successeursVdm

Ajouter pointdepartHarry à la fin de la liste cheminHarry
Ajouter pointdepartVdm à la fin de la liste cheminVdm
Si armeHarry n'est pas NIL et armeHarry n'a pas déjà été ramassée par Harry ou Voldemort alors
    Ajouter arme dans armespossedeesHarry
Fin SI
Si armeVdm n'est pas NIL et armeVdm n'a pas déjà été ramassée par Harry ou Voldemort et pointdepartVdm n'est pas dans cheminVdm avant d'avoir ajouté pointdepartVdm alors
    Ajouter arme dans armespossedeesVdm
Fin SI
(Ces deux blocs conditionnels permettent la détection d'une potentielle arme sur la case actuelle pour les 2 personnages et s'il elle n'a pas été ramassé avant ou si le personnage n'y est pas déjà passé (inutile pour Harry car il ne repasse jamais au même endroit sauf en remontant mais il n'exécute pas toutes les actions alors))
Si pointdepartHarry= pointdepartVdm et armespossedeesVdm contient le Sortilege de la mort alors
    Ajouter Harry Potter dans horcruxedetruitVdm
    **Retourner NIL**
Sinon
    Si horcruxeHarry n'est pas NIL et n'est pas dans horcruxedetruitHarry ou n'est pas dans horcruxedetruitVdm alors
        Si le résultat de la fonction hasBonneArme avec les paramètres horcruxeHarry, armespossedeesHarry et horcruxesDescription n'est pas NIL alors

Ajouter horcruxeHarry dans horcruxedetruitHarry
Fin Si
Fin SI
Si horcruxeVdm n'est pas NIL et n'est pas dans horcruxedetruitHarry ou n'est pas dans horcruxedetruitVdm et pointdepartVdm n'est pas dans cheminVdm avant d'avoir ajouté pointdepartVdm alors
Si le résultat de la fonction hasBonneArme avec les paramètres horcruxeVdm, armespossedeesVdm et horcruxesDescription n'est pas NIL alors
Ajouter horcruxeVdm dans horcruxedetruitVdm
Fin Si
Fin SI
(Ces 2 blocs permettent la détection d'un potentiel Horcruxe sur la case actuelle pour chaque personnage)
Si profondeurMax != 0 alors
Si successeursHarry n'est pas NIL
Pour chaque position succ de successeursHarry
Si succ appartient au résultat de l'appel de successeurs-valides avec les paramètres pointdepartHarry, carte et cheminHarry (cette condition est nécessaire dans le cas d'un croisement des branches par la suite pour éviter de retourner sur des case déjà parcourues)
Lire l'entrée de l'utilisateur en console et stocker le résultat dans caseVdm
Si caseVdm n'est pas un nombre ou n'est pas dans successeursVdm
Attribuer le premier élément de successeursVdm dans caseVdm
Fin Si
Attribuer à brancheActuelle le résultat de l'appel de la fonction actuelle exploreprofondeurVoldemort avec les paramètres carte horcruxesDescription, horcruxesMap, armesMap, profondeurMax-1, succ, caseVdm, cheminHarry, cheminVdm, horcruxedetruitHarry, horcruxedetruitVdm, armespossedeesHarry et armespossedeesVdm
Si brancheActuelle vaut NIL (Harry est mort dans la branche)
Attribuer 0 à brancheActuelle
Sortir de la boucle Pour
Fin Si
Modifier horcruxedetruitHarry avec la valeur correspondante dans brancheActuelle

Modifier armespossedeesHarry avec la valeur correspondante dans brancheActuelle
Modifier cheminHarry avec la valeur correspondante dans brancheActuelle et ajouter la valeur de pointdepartHarry (pour que le chemin contienne le vrai parcourt sans « sauts »)
Modifier horcruxedetruitVdm avec la valeur correspondante dans brancheActuelle
Modifier armespossedeesVdm avec la valeur correspondante dans brancheActuelle
Modifier cheminVdm avec la valeur correspondante dans brancheActuelle et ajouter la valeur de pointdepartVdm (pour que le chemin contienne le vrai parcourt sans « sauts »)
Modifier successeursVdm avec la valeur renvoyé par l'appel de la fonction successeurs-valides avec les paramètres : dernier élément de cheminVdm (correspond à la dernière position de Vdm dans la branche), carte et NIL (car Voldemort n'a pas de restrictions de déplacement parmi les cases adjacentes)

Fin Si

Fin Pour

Fin Si

Fin Si

(Ce bloc permet le déplacement de Voldemort et d'Harry et de parcourir toutes les branches si il y a plusieurs fils au potentiel père actuel (décrémentation de 1 en profondeur). Puis elle permet de modifier les variables qui étaient les paramètres optionnels pour les retourner avec les bonnes valeurs ajoutées par le parcourt)
Si brancheActuelle=0 (ce n'est pas une liste donc Harry est mort entre temps, on a choisi 0 arbitrairement mais on aurait pu juste tester si c'est NIL et ne jamais lui attribuer 0)

**Retourner NIL**

Sinon

**Retourner la liste contenant horcruxedetruitHarry, armespossedeesHarry, cheminHarry, horcruxedetruitVdm, armespossedeesVdm et cheminVdm** (pour l'implémentation on utilisera des clés pour pouvoir y accéder plus facilement).

Fin Si

Fin Si

## Code Lisp

Le code Lisp associé à l'algorithme précédent étant relativement long, il est disponible dans le fichier avec l'extension .lisp de l'archive.

# VI. Créativité

Cette version permet de rendre la première version plus en adéquation avec l'œuvre de J.K. Rowling.
Par conséquent, Voldemort n'a plus l'objectif de détruire les Horcruxes, les reliques de la mort ont été rajoutées et des Détraqueurs sont désormais présents.
Nous allons expliquer tous ces ajouts en détails.

## Ajouts des reliques de la mort

Dans l'univers de Harry Potter, les reliques de la mort sont des objets puissants qui ont des pouvoirs spécifiques. Ces reliques sont au nombre de 3 : La pierre de résurrection, La baguette de sureau, et la cape d'invisibilité.

Voici des extrais du Wiki de Harry Potter sur ces différentes reliques :
https://harrypotter.fandom.com/fr

### *La pierre de résurrection*

"Contrairement à ce que le nom de Pierre de Résurrection pourrait faire croire, les morts invoqués ne sont pas réellement ressuscités. Ils apparaissent en tant que souvenir dont les contours sont légèrement flous, moins consistant que de véritables êtres vivants mais plus que des fantômes"

Nous avons donc décider de faire agir la pierre de résurrection de la manière suivante : Seul Harry peut récupérer la pierre. Dans le cas ou Harry récupère la pierre, il fait apparaitre le fantôme de ses parents qui lui indique l'emplacement d'un horcruxe. Harry peut alors détruire cet Horcruxe instantanément s'il possède la méthode de destruction. Par contre, ses parents ne savent pas si l'Horcruxe a déjà été détruit ou non, ils pourront alors lui indiquer des emplacements où il n'y a plus d'Horcruxes. Par contre, ce pouvoir est utilisable à chaque tour et permet (avec de la chance) de tomber sur des anciens Horcruxes que Harry n'avait pu détruire, faute d'avoir la bonne arme, pour ainsi retenter sa chance.

### *La baguette de sureau*

"Voldemort tente de tuer Harry en lui lançant un sortilège de Mort qui est contré par un sortilège de Désarmement lancer par son adversaire. La Baguette de Sureau retourne alors l'*Avada Kedavra* contre son lanceur qui lâche alors la Baguette que Harry attrape avec un geste vif, la Baguette de Sureau étant enfin dans les mains de son véritable maître."

Ainsi, nous avons fait le choix suivant : La baguette de sureau permet à Harry Potter, s'il se trouve sur la même case que Voldemort et que Voldemort possède "Sortilège de Mort" et "Baguette de sureau" de tuer Voldemort. On considère ici que le sortilège c'est retourné contre son propriétaire.

De plus on considère que la Baguette de sureau n'est récupérable que par Voldemort.

***La cape d'invisibilité***

"Les enchantements qui recouvrent la Cape sont très puissants. Elle est capable de résister à n'importe quel type de sort qui pourrait endommager ou rendre inefficace une cape d'invisibilité normale et est probablement ensorcelée pour être à l'épreuve du temps."

Ainsi, on comprend que rien ne permet à Voldemort de voir Harry à travers cette cape. On considère donc que la cape d'invisibilité permet à Harry de ne pas être détecté par Voldemort même s'ils sont sur la même case. Dans le cas ou Harry et Voldemort sont sur la même case, il ne se passe donc rien, si Harry a cette cape.
Enfin, la cape d'invisibilité n'est récupérable que par Harry Potter.

## Fonctions utilitaires

Nous avons pu constater que les fonctions écrites dans les parties III et IV comportent déjà une grande quantité de lignes. L'objectif est donc de créer plus de fonctions utilitaires afin de rendre plus lisible le code et pouvoir ajouter d'autre fonctionnalité par la suite.

***Fonction estici***

Renvoie la liste des éléments présent sur la case d'après la description Mapdescription donnée.
Par exemple, estici (8 horcruxesMap) renvoie ("Journal intime de Tom Jedusor")

L'intérêt de cette fonction est qu'elle fonctionne aussi bien pour tester des armes, que des Horcurxes ou encore des reliques.

```
; estici (int list(list)) -> list

(defun estici (point Mapdescription)
        (cdr (assoc point Mapdescription)) ; attention c'est une liste
)
```

***Fonction nondejarecup***

Renvoie True si l'objet donné en argument n'est pas dans les listes : "ListeObjetPerso1" et "ListeObjetPerso2" et nil sinon.
Encore une fois cette fonction peut être utilisée pour vérifier la récupération d'Horcruxes, d'armes ou de reliques en adaptant les arguments.
Concernant le fonctionnement du programme, on utilise uniquement un if.

```
; nondejarecup (char list list) -> bool
; exemple : (nondejarecup "Journal de Tom Jedusor" nil  nil) renvoie true
```

```
(defun nondejarecup (Objet ListeObjetPerso1 ListeObjetPerso2)
        (if (and (not (member Objet ListeObjetPerso1 :test #'string=)) (not (member Objet
ListeObjetPerso2 :test #'string=)))
                T
                nil
        )
)
```

***Fonction testearme***

Renvoie la liste des armes possédées par un personnage après qu'il ait récupéré si possible les armes sur la case "point".
Ainsi, effectuer (setq ArmesPersoActif (testearme point ArmePersoActif ArmePersoAutre armesMap) revient à tester les armes de la case point et faire récupérer ces armes au personnage actif.
On a egalement fait en sorte que toutes les fonction utilitaires renvoie leurs propres phrases d'information afin d'encompbrer la fonction principale avec des « format t « … » »

Dans l'ordre, la fonction effectue les actions suivantes :
Récupérer la liste de toute les armes sur la case point en utilisant la fonction "estici"
S'il y a des armes, pour toutes les armes récupérées,
On vérifie que ces armes n'ont pas déjà été récupéré avec "nondejarecup"
On ajoute les armes non récupérées à ArmesPersoActif
On renvoie la liste ArmesPersoActif.

```
;testearme (int, list, list, list(list)) -> list
;exemple : testearme (25 nil nil armesMap) renvoie ("EpÃ©e de Gryffondor")

(defun testearme(point ArmePersoActif ArmePersoAutre armesMap)
(let ((listearme (estici point armesMap)))

        (if  listearme ;si la liste des armes est non vide
                (dolist (arme listearme)
                        (if (nondejarecup arme ArmePersoActif ArmePersoAutre)
                                (progn
                                        (format t "On a trouvé l'arme : ~a ! ~%" arme)
                                        (setq ArmePersoActif (append (list arme)
                                        ArmePersoActif)) ;si la/les armes sont encore sur la case,
                                        on ajoute.
                                        )
                                )
                        )
                )
                (format t "Il n'y a pas d'arme sur la case : ~a . ~%" point)
        )
```

```
        (format t "Le personnage a en sa possession les armes : ~a . ~%" ArmePersoActif)
        ArmePersoActif ;dans tous les cas on renvoie la liste des armes possédées
)
```

### *Fonction testehorcruxe*

La fonction “testehorcruxe” effectue les mêmes actions que “testearme” mais pour les horcruxes, elle prend donc également en compte la condition d’arme et de chemin : il faut avoir la bonne arme et arriver pour la première fois sur la case.

La fonction fonctionne exactement comme la fonction précédente avec des tests en plus :
Un teste utilisant hasbonnearme pour vérifier que le personnage possède bien l’arme nécessaire a la destruction de l’Horcruxe.
Un teste sur le chemin pour vérifier que le personnage n’est pas déjà passé sur la case.

```
;testehorcruxe (int, list, list, list) -> list
;Les prints nécessaires sont déjà inclus dans la fonction

(defun testehorcruxe (point armespossedees horcruxedetruit_p1 horcruxedetruit_p2 chemin horcruxesMap)
(let ((horcruxe (estici point horcruxesMap)))
        (if horcruxe
                (dolist (h horcruxe)
                        (if (nondejarecup h horcruxedetruit_p1 horcruxedetruit_p2)
                                (progn
                                        (format t "Il y a l'Horcruxe ~a sur cette case ! ~%" h)
                                        (if (and (hasBonneArme h armespossedees horcruxesDescription) (not (member point chemin)))
                                                (progn
                                                        (setq horcruxedetruit_p1 (append horcruxedetruit_p1 (list h)))
                                                        (format t "Le personnage a detruit l'Horcruxe ~a ! ~%" h)
                                                )
                                                (format t "Le personnage ne peut pas detruire l'Horcruxe ~a ! ~%" h)
                                        )
                                )
                        )
                )
        )
)
(format t "Pour l'instant, le personnage a détruit les horcruxes : ~a . ~% " horcruxedetruit_p1)
horcruxedetruit_p1
)
```

### *Fonction testereliques*

Teste en fonction du choix du personnage : 0 pour Harry et 1 pour Voldemort, les reliques récupérables par le personnage sur la case point. La fonction renvoie alors les reliques possédées par le personnage avec l'ajout éventuel de reliques.

Cette fonction fonctionne dans les grandes lignes comme la fonction "testearme" mais teste uniquement certaine relique en fonction de la valeur de la valirable "perso".

```
;testerelique (int list list int) -> list

(defun testerelique (point reliquespossedees reliqueMap perso) ;0 :Harry et 1 :Vdm
(let ((reliques (estici point reliqueMap)))
        (if reliques
                (dolist (r reliques)
                        (if (and (or (and (string= r "Pierre de résurrection") (nondejarecup "Pierre
                de résurrection" reliquespossedees nil)) (and (string= r "Cape d'invisibilite")
                (nondejarecup "Cape d'invisibilite" reliquespossedees nil))) (eql perso 0))
;teste de la cape et de la pierre de résurrection uniquement pour Harry Potter
                                (progn
                                        (format t "Harry Potter a trouvé la relique : ~a ! ~%" r)
                                        (setq reliquespossedees (append (list r)
                                reliquespossedees))
                                        (format t "Harry a en sa possession les reliques : ~a . ~%"
                                reliquespossedees)
                                )
                        )
                        (if (and (string= r "Baguette de sureau") (nondejarecup "Baguette de
                sureau" reliquespossedees nil) (eql perso 1)) ;La baguette, uniquement pour
                Voldemort
                                (progn
                                        (format t "Voldemort a trouvé la relique : ~a ! ~%" r)
                                        (setq reliquespossedees (append (list r)
                                reliquespossedees))
                                        (format t "Voldemort a en sa possession les reliques : ~a .
                                ~%" reliquespossedees)
                                )
                        )
                )
        )
)

(if (not (or (eql perso 0) (eql perso 1))) ;ligne d'erreur
        (format t "Argument perso invalide. Choisir 0 pour Harry ou 1 pour Voldemort")
reliquespossedees ;si pas d'erreur on renvoie la liste des armes possédées
)
)
```

### *Fonction interaction_Harry_Vdm :*

Prend en argument la position de Harry “posHarry”, celle de Voldemort “PosVdm”, les reliques de Harry “reliqueHarry”, celle de Voldemort “reliqueVdm” et les armes de Voldemort “armeVdm”.

Renvoie True si un personnage est tué et Nil sinon. Dans tous les cas, la fonction imprimera des phrases d’explication pour que l’utilisateur comprenne  reliqueVdm l’interaction qui a lieu.

Concernant le fonctionnement de la fonction, c’est uniquement une serie de “if” et de “format” pour faire tous les cas en fonction des reliques et armes possédées par les deux joueurs.

La fonction étant assez longue, nous vous invitons à regarder le fichier. lisp directement.

Voici quelques cas de situation possibles en langage naturel :

SI Voldemort a la baguette de sureau et le sortilège de Mort, et que Harry n’a pas la cape d’invisibilité. Voldemort meure.

SI Voldemort n’a pas la baguette de sureau et le sortilège de Mort, et que Harry n’a pas la cape d’invisibilité. Harry meure.

Si Harry a la cape d’invisibilité, il ne se passe rien dans tous les cas.

Si Voldemort n’a pas le sortilège de Mort, il ne se passe rien dans tous les cas.

## Ajouts des détraqueurs

### *Description de l’ajout*

Dans l’univers d’Harry Potter, les détraqueurs sont des créatures maléfiques extrêmement dangereuses. Le Wiki Harry Potter dit à leurs sujets :
“Un Détraqueur est une créature des ténèbres considérée comme la plus abjecte qui soit au monde. Les Détraqueurs se nourrissent de la joie humaine, et provoquent par la même occasion du désespoir et de la tristesse sur quiconque se trouve à proximité. Ils sont aussi capables d'aspirer l'âme d'une personne, laissant leur victime dans un état végétatif irréversible. Il n'existe qu'un seul moyen de les repousser, le sortilège du Patronus.”

Nous avons fait le choix d’ajouter ces créatures à nos programmes. Le nombre de détraqueurs est fixé par l’utilisateur de la fonction à un nombre inférieur au nombre de cases sur lesquelles les personnages peuvent se déplacer. Les détraqueurs apparaissent sur des cases différentes et aléatoires. Par la suite les détraqueurs se déplacent à chaque tour sur des cases aléatoires (en se téléportant). Ces derniers n’affectent que Harry Potter. Quand Harry Potter se trouve sur la même case qu’un détraqueur, il perd un tour le temps de lancer un sort de “Patronus”. Pendant ce temps, Voldemort continue à agir normalement (à noter que Voldemort ne peut probablement pas lancer de sort de Patronus car ce dernier nécessite d’être remplit d’émotions positives). Il n’est donc pas affecté par les détraqueurs. Lorsque Harry a lancé un

"Patronus", le détraqueur est détruit, il y a donc un détraqueur en moins qui se déplace sur la carte.

### *Fonction initialisedetraqueurs*

Cette fonction prend en argument un nombre de détraqueurs "nombre" et la carte. Elle renvoie la liste de la position de "nombre" détraqueurs, sur des cases valides (ou l'on peut se déplacer), aléatoire et différentes pour chaque détraqueur (de sorte que 2 détraqueurs ne se trouvent jamais au même endroit sur la carte).

Tout d'abord on teste la validité des arguments, c'est à dire que nombre est bien un entier et carte une liste.
On récupère ensuite tous les point de toutes les listes de "carte". Il faut alors supprimer les doublons avec "remove-duplicate" pour obtenir une liste de tous les points visitables de la carte.
On vérifie que le nombre de détraqueurs demandé est inférieur au nombre de cases disponibles. Si ce n'est pas le cas on renvoie un message d'erreur qui précise le nombre maximal de détraqueurs admis.
Par la suite, on tire "nombre" case parmi les cases récupérées aléatoirement grâce à la fonction "random" (qui tire un nombre aléatoire entre 0 et un entier donné) et la fonction "elt" (qui permet d'accéder à un élément d'indice donné dans une liste). A chaque fois qu'une case est tirée, elle est supprimée de la liste pour s'assurer que la position de chaque détraqueur est différente.

```
;intialisedetraqueurs (int list) -> list

(defun initialisedetraqueurs (nombre carte)
(if (and (numberp nombre) (listp carte))
        (let ((listeptslibres nil) (listedetraqueurs nil))
                (progn
                        (dolist (listepoint carte) ;recupere tout les points visitables avec 1 seule occurence.
                                (dolist (point carte)
                                        (setq listeptslibres (append point listeptslibres))
                                )
                        )
                        (setq listeptslibres (remove-duplicates listeptslibres :test #'eql))
                        (if (<= nombre (length listeptslibres)) ;teste de la validité du nombre donnée en argument

                        (dotimes (i nombre)  ;on tire les n detraqueurs parmis les cases valides
                                        (progn
                                                (setq listedetraqueurs (append (list (elt listeptslibres (random (length listeptslibres)))) listedetraqueurs)) ;selection un element de listeptslibres et l'ajoute Ã  listedetraqueurs
```

```
                                (delete (car listedetraqueurs) listeptslibres :test
                            #'eql) ;suprime l'element selectionne pour eviter de le tirer
                            2 fois.
                                )
                            )
                            (format t "~%Erreur : Le nombre de detraqueurs est plus
                        grand que le nombre de case visitable. Veuillez choisir un nombre
                        plus petit que ~a . ~%" (length listeptslibres))
                        )
                        listedetraqueurs
                    )

                )
        (format t "~%Erreur dans un type d'entree"))
)
```

### *Fonction detraqueurici*

Une fonction extrêmement courte qui prend argument la liste des positions des détraqueurs et un point et renvoie True s'il y a un detraqueur sur la case “point” et l’indique à l’utilisateur de la fonction. Et nil sinon.

```
;detraqueurici (int list) -> bool
;exemple : (detraqueurici 10 (10 5 7)) renvoie True

(defun detraqueurici (point detraqueursMap)
        (if (member point detraqueursMap)
                (progn
                        (format t "Harry rencontre des detraqueurs !!! Il passe son tour pour
                lancer un sort de patronus~%")
                        T
                )
                nil
        )
)
```

## Fonction Principale

Il est maintenant temps d’implémenter une unique fonction reprenant toutes les fonctionnalités décrites précédents. Cette fonction est particulièrement imposante, nous ne la mettons donc pas dans le rapport. Nous allons expliquer le fonctionnement de la fonction :

### *Arguments*

**Carte** (list) : la carte sur laquelle se deplace les personnages

**HocruxesDescription** (list) : contient les méthodes de destruction de chaques Horcruxes

**HorcuxesMap, ArmesMap, reliquesMap** (list) : contiennent respectivement la position des horcruxes, des armes et des reliques sur la carte.

**ProfondeurMax** (int) la profondeur max de l'exploration en profondeur.

**PosHarry, PosVdm** (int) : sont les positions de depart de Harry Potter et Voldemort

**Nombredetraqueurs** (int) : nombre de detraqueurs au debut du jeu sur la carte.

Arguments optionnels :

**CheminHarry cheminVdm** (list) : les chemins respectifs de Harry et Voldemort

**horcruxeHarry horcruxeVdm armesHarry armesVdm reliquesHarry reliquesVdm** (list) respectivement les Horcuxes, armes et reliques recupéré par Harry et Voldemort.

***Algorithme***

Voici le début de l'algorithme de la fonction en langage naturel

Générer la position des détraqueurs dans "emplacementDetraqueurs"

Si Harry n'est pas sur un détraqueur :

Ajouter PosHarry dans cheminHarry

Ajouter PosVdm dans CheminVdm

Si Harry n'est pas sur un détraqueur :

Tester les armes de Harry (Harry recupère les armes)

Tester les armes de Vdm dans tous les cas

Vérifier l'interaction entre Harry et Vdm, agir en conséquence.

Si interaction : renvoyer les chemins, armes, reliques, Horcurxes des deux personnages

La suite et fin du programme fonctionne comme la fonction d'exploration en profondeur de Harry et Voldemort. Mais effectue également la récupération des

reliques de la mort. Enfin Harry ne se déplace que si Harry n'est pas sur un détraqueur.

# VII. Conclusion

Ce TP02 a été encore une fois très enrichissant, le sujet se basais sur une case plus concrète que d'habitude à travers l'univers de Harry Potter, ce qui nous a permis de nous concentrer plus facilement sur le code à produire. Ce sujet à contribué à l'amélioration de notre niveau de programmation en lisp notamment pour les recherches dans les espaces d'état, mais également la manière générale par le biais de la partie créativité. Pour la partie créativité nous avons en effet été très inspiré, ce qui nous a parmi d'ajouter des fonctionnalité intéressantes et assez complexes par rapport au sujet de départ. Nous avons néanmoins dû faire attention au temps qui nous était accordé pour le TP afin de produire un code fonctionnel dans les temps. Nous avons également dû réfléchir à la manière de découper le code en des fonction utilitaires afin de rendre la fonction principale lisible. Dans tous les cas, cette partie créativité à pleinement contribué à l'amélioration de notre niveau de programmations en nous poussant à produire de code plus complexe.

Concernant les améliorations possibles et qui n'ont pas été traitées dans le sujet, nous aurions pu discuter également de la recherche en profondeur optimisée pour que Harry récupère un maximum d'armes et d'Horcruxes en un minimum de déplacement (ou du moins en un minimum de profondeur), si une telle solution existe. Pour ce faire, nous aurions pu calculer tous les chemins possibles et sélectionner le meilleur, seulement, la complexité en temps d'un tel programme est assez importante. Un tel programme aurait surement fonctionné pour une carte de la taille de l'énoncé, mais certainement pas pour des cartes comptants plus de cases.

Concernant les difficultés rencontrées lors de ce TP, elle se sont concentrées sur les fonctions de parcours en profondeurs pour Harry, Harry et Voldemort, et la dernière de la partie créativité. Nous avons également cherché le moyen de présenter du code sur ce rapport (fait via le traitement de texte Word) Nous avons d'abord cherché à faire des captures d'écran pour conserver les couleurs, seulement les codes étaient difficilement lisibles. La meilleure solution que nous avons trouvée et de convertir nos codes en fichier RTF pour conserver les couleur et l'indentation, puis de copier-coller sur la présentation. Pour rappel, tous les codes sont lisibles et testable dans le document .lisp.

Nous vous remercions pour votre lecture.