

CAVELIUS Walid  
DERDAKI Tareq



**TP02 : Harry Potter et les reliques de la mort, partie 3**

# Sommaire

<u>Sommaire</u>	1
<u>Introduction</u>	2
<u>Analyse</u>	3
<u>Arbres de Recherche en Profondeur</u>	3
<u>A - Départ Case 1</u>	4
<u>A - Départ Case 36</u>	5
<u>Fonctions de service</u>	6
<u>AJOUTER RAPPEL DE LA CARTE ETC</u>	6
<u>Fonction successeurs-valides</u>	6
<u>Recherche en Profondeur pour la recherche de Harry Potter</u>	9
<u>1 - Algorithme de Recherche en Profondeur</u>	9
<u>2 - Code LISP</u>	11
<u>1ère Partie : Affichage de la position et Ajout de la case au chemin</u>	12
<u>2ème Partie : Arme dans la case ?</u>	12
<u>3ème Partie : Horcruxe dans la case ?</u>	13
<u>4ème Partie : études des successeurs valides</u>	13
<u>5ème Partie : Renvoie des données actualisées</u>	14
<u>Lord Voldemort part à la recherche des Horcruxes</u>	15
<u>1) Algorithme précédent modifié</u>	15
<u>2) Code LISP</u>	17
<u>Partie 1 : Initialisation et Conditions de Fin</u>	17
<u>Partie 2 et 3 : Actions pour Harry et Voldemort</u>	17
<u>Partie 4 : Exploration des Successeurs</u>	19
<u>Partie 5 : Renvoi des Données Actualisées</u>	19
<u>Créativité</u>	20
<u>1. Représentation visuelle améliorée en ASCII</u>	20
<u>2. Mise en place d'énigmes pour ramasser des armes</u>	21
<u>3. Mode multijoueur</u>	21
<u>Mode Multijoueur Stratégique : Conflit Entre Sorciers</u>	21
<u>Conclusion</u>	23

# Introduction

Lors de ce TP, dont l'énoncé est basé sur le lore d'Harry Potter, nous mettrons en pratique nos connaissances en recherche dans un espace d'états. En effet, notre devoir sera grossièrement de mettre en place (en LISP) une sorte de chasse au trésor (ou de course plus tard), dans laquelle il faudra retrouver, puis détruire les Horcruxes cachés sur la carte. Cela se fera au moyen de méthodes de destruction (elles aussi cachées sur la carte) qu'il faudra retrouver au préalable. Pour ce faire, nous découperons notre travail en un nombre conséquent de fonctions (ou sous-fonctions) afin de le rendre plus lisible et propre.

# Analyse

## Arbres de Recherche en Profondeur

Dans cette question, nous dessinerons les arbres de recherche en profondeur au départ de la case 1 et de la case 36. Bien évidemment, cela sera réalisé conformément aux règles imposées par l'énoncé. L'espace d'états est le suivant (cf. Schéma 1 ci-dessous) :

1	12	13	24	25	36
2	11	14	23	26	35
3	10	15	22	27	34
4	9	16	21	28	33
5	8	17	20	29	32
6	7	18	19	30	31

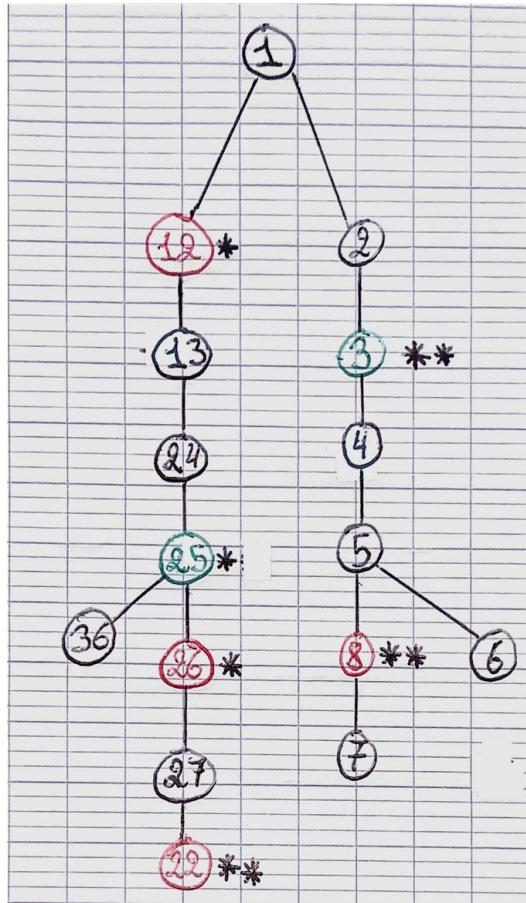
Schéma 1 : Espace d'états

Les états représentés en vert sur les arbres correspondent aux états cachant une méthode de destruction. Il en va de même pour les cases rouges et les Horcruxes. De plus, si le même pictogramme est présent aux côtés de plusieurs nœuds, cela signifie que les horcruxes présentes dans ces cases sont détruites par la même méthode de destruction (dont le nœud sera aussi représenté avec ce même pictogramme).

Avant de démarrer, rappelons que la profondeur maximale de recherche est imposée par l'énoncé et qu'elle est de 7 et que nous explorons les cases voisines dans l'ordre suivant : Haut-Droite-Bas-Gauche.

## A - Départ Case 1

L'arbre de recherche en profondeur pour le départ de la **case 1** est le suivant (cf. Arbre 1) :



Arbre 1 : Départ Case 1

Le parcours de l'arbre est le suivant (les remontées aux noeuds parents sont implicites) :

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

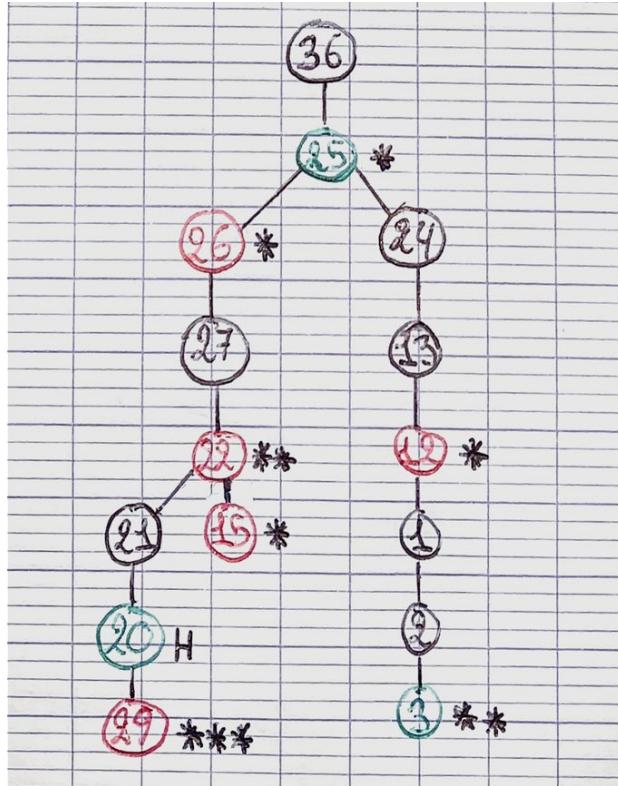
En démarrant à la **case 1**, nous pouvons récupérer les horcruxes suivantes :

- **Journal Intime de Tom Jedusor** (case 8)
- **Nagini** (case 26)

En augmentant la profondeur de la recherche, on pourrait détruire la **Bague de Gaunt** (case 15) mais on ne pourrait pas détruire le **Médaillon de Salazar Serpentard** (case 12), la **Coupe de Helga Poufsouffle** (case 22) et le **Diadème de Rowena Serdaigle** (29). En effet, pour détruire le **Médaillon de Salazar Serpentard** (respectivement le **Diadème de Rowena Serdaigle** et la **Coupe de Helga Poufsouffle** ), il nous faudrait la méthode de destruction qui se cache à la **case 25** (respectivement **case 32** et **case 3**). Cependant, il est impossible de se rendre à ces cases sans passer d'abord par les Horcruxes sans méthode de destruction. Cela aura pour conséquence qu'on ne pourra plus jamais détruire ces horcruxes. Ainsi, il nous est **impossible** de détruire toutes les horcruxes en démarrant de la case 1, et ce, même en ne posant aucune limite sur la profondeur.

## A - Départ Case 36

L'arbre de recherche en profondeur pour le départ de la **case 36** est le suivant (cf. Arbre 2) :



Arbre 2 : Départ Case 36

Le parcours de l'arbre est le suivant :

36 - 25 - 26 - 27 - 22 - 21 - 20 - 29 - 15 - 24 - 13 - 12 - 1 - 2 - 3

En démarrant à la **case 36**, nous pouvons les horcruxes suivantes :

- **Médaillon de Salazar Serpentard** (case 12)
- **Nagini** (case 26)
- **Bague de Gaunt** (case 15)
- **Coupe de Helga Poufsouffle** (case 22)

En augmentant la profondeur de la recherche, on pourrait détruire la **Journal Intime de Tom Jedusor** (case 8) mais on ne pourrait pas détruire le **Diadème de Rowena Serdaigle** (case 29) et la **Coupe de Helga Poufsouffle** (case 22). En effet, pour détruire le **Diadème de Rowena Serdaigle**, il nous faudrait la méthode de destruction qui se cache à la **case 32**. Cependant, il est impossible de se rendre à cette case sans passer d'abord par l'Horcruxe (et sans la méthode de destruction). Cela aura pour conséquence qu'on ne pourra plus jamais détruire cette horcruxe (même raisonnement pour la **Coupe de Helga Poufsouffle**). Ainsi, il nous est **impossible** (encore une fois) de détruire toutes les horcruxes en démarrant de la case 36, et ce, même en ne posant aucune limite sur la profondeur.

# Fonctions de service

## Fonction **successeurs-valides**

Cette fonction prend en argument une case, une carte et le chemin parcouru puis renvoie les successeurs (de la case) qui n'ont pas encore été explorés/parcourus. Pour faciliter la compréhension (ainsi que l'application) et rendre le code le plus lisible, nous avons décidé de créer une sous-fonction **successeurs** qui renvoie simplement les successeurs d'une case qu'elle prend en paramètre :

```
(defun successeurs (case carte)
  (cdr (assoc case carte)))
```

**Raisonnement algorithmique** : On retrouve la case dans la carte grâce à **assoc** et on renvoie uniquement ses successeurs avec un **cdr**. A titre **d'exemple** :

```
(successeurs '1 map)      donne      CG-USER(3):  
                                (12 2)
```

Pour ce qui est de la fonction **successeurs-valides**, elle est définie de la manière suivante :

```
(defun successeurs-valides (case carte chemin_parcouru)
  (let
    ((cases_suivantes_valides nil) (cases_suivantes (successeurs case carte)))
    (dolist (case_actuelle cases_suivantes cases_suivantes_valides)
      (if (not (member case_actuelle chemin_parcouru))
          (push case_actuelle cases_suivantes_valides)))))
```

**Raisonnement algorithmique** : D'abord, on crée une variable locale **cases\_suivantes** (qui **récupère tous les successeurs** de la case passée en paramètre grâce à un appel sur la fonction **successeurs**) et une variable **cases\_suivantes\_valides** que l'on initialise à **nil**. Une fois que cela est fait, nous parcourons chaque case de la liste **cases\_suivantes** et nous testons si ces derniers font partie du **chemin\_parcouru** (passé en paramètre). Si c'est le cas, rien ne se passe. Si ce n'est pas le cas, nous ajoutons la case à la liste **cases\_suivantes\_valides** que nous renverrons à la fin de la fonction, une fois que tous les successeurs valides potentiels auront été testés. A titre **d'exemple** :

```
(successeurs-valides '1 map '(3 4 5 12)) donne CG-USER(5):  
                                (2)
```

## 2 - Fonction methodeDestruction

La fonction reçoit en argument un **Horcruxe** (en vérité son nom) ainsi que la **liste de description des Horcruxes** comme décrite par l'énoncé. La fonction **methodeDestruction** renvoie la méthode de destruction de l'Horcruxe si cette dernière est répertoriée et nil sinon.

```
(defun methodeDestruction (Horcruxe HorcruxesDescription)
  (dolist (horcruxe_actuelle HorcruxesDescription)
    (if (equal Horcruxe (car horcruxe_actuelle))
        (return-from methodeDestruction (cadr(cadr horcruxe_actuelle)))
    )))
```

**Raisonnement algorithmique** : la fonction parcourt chaque description d'Horcruxe de la liste de description et réalise un test d'égalité sur le car chaque description (ce qui correspond au nom de l'Horcruxe) et l'Horcruxe passé en paramètre. Si ce test est vérifié, alors nous retournons simplement le cadr du cadr de la description qui correspond au nom de la méthode de destruction associée. Si aucun test n'est vérifié, la fonction renvoit nil. A titre d'exemple :

```
(methodeDestruction "Nagini" horcruxesDescription)
```

donne

```
CG-USER(7):
"Epée de Gryffondor"
```

Ou encore :

```
(methodeDestruction "IA01" horcruxesDescription)
```

donne évidemment

```
CG-USER(8):
NIL
```

### 3 - Fonction hasBonneArme

La fonction prend en argument un **Horcruxe**, la liste des méthodes de destruction possédées ainsi que la liste de description des Horcruxes. La fonction **hasBonneArme** renvoie **T** si la méthode de destruction associée à l'Horcruxe passée en paramètre est contenue dans la liste des méthodes de destruction et renvoie **nil** sinon.

```
(defun hasBonneArme (horcruxe horcruxesDescription arsenal)
  (let ((methode (methodeDestruction horcruxe horcruxesDescription))
        (armePossedee))
    (dolist (arme arsenal)
      (if (equal arme methode)
          (setq armePossedee T)))
    armePossedee))
```

**Raisonnement algorithmique** : Tout d'abord, nous créons des variables locales **methode** (qui reçoit la **méthode de destruction associée à l'Horcruxe** passée en paramètre) et **armePossedee** (qui, au départ est égale à **nil** mais qui pourrait prendre la valeur **T** par la suite). Une fois cela fait, nous parcourons chaque arme (ou **méthode de destruction**) présente dans la **liste des armes possédées** (que nous avons ici nommée **arsenal**) et réalisons un **test d'égalité** entre l'arme parcourue et **methode**. Si ce test est vérifié, alors **armePossedee** prend la valeur **T**, sinon, rien ne se passe. A la fin, si la méthode adéquate est présente dans la liste des armes possédées, alors la fonction renvoie **T** et **nil** sinon. A titre d'exemple :

```
(hasBonneArme '"Nagini" horcruxesDescription '("Epée de Gryffondor" "Sortilège de la Mort"))
                                                     donne
CG-USER(9):
T
```

ou encore :

```
(hasBonneArme '"Nagini" horcruxesDescription '("Sortilège de la Mort"))
                                                     donne
CG-USER(10):
NIL
```

# Recherche en Profondeur pour la recherche de Harry Potter

Dans cette partie du TP, nous devrons réaliser un **algorithme** puis un **code LISP** permettant “d’automatiser” une recherche en profondeur (toujours de **profondeur maximale 7**), et de renvoyer (entre autres) les **armes collectées** et les **horcruxes détruites** au cours du parcours. Dans le **code LISP**, nous veillerons également à afficher sa **position** à chaque **déplacement**, l'**inventaire de ses armes** à chaque **ajout** et la **liste des horcruxes détruites** à chaque nouvelle destruction.

## 1 - Algorithme de Recherche en Profondeur

```
def Parcours_Harry ( case carte profondeur HorcruxesDescription
                      HorcruxesDétruites / HorcruxesTop AmesTop
                      arsenal cheminParcouru )
    Ajouter case à cheminParcouru
    Si Ame présente dans case
        Ajouter Ame à arsenal
        Supprimer Ame de AmesTop
    Si Horcruxe présente dans case
        Si hors Bonne Horcruxe
            Ajouter Horcruxe à HorcruxesDétruites
            Supprimer Horcruxe de HorcruxesTop
        Si profondeur < 7
            Init cases_suivantes (successives-valides case carte )
            Pour chaque case dans cases_suivantes
                (setq data Parcours_Harry ( case carte (+ profondeur 1)
                                              HorcruxesDescription HorcruxesDétruites
                                              HorcruxesTop AmesTop arsenal
                                              cheminParcouru ))
                (setq HorcruxesTop (car data))
                (setq HorcruxesDétruites (cadr data))
                (setq AmesTop (caddr data))
                (setq arsenal (cdddr data))
                (setq cheminParcouru (cadf (cddddr data)))
            Retourne (list HorcruxesTop HorcruxesDétruites AmesTop
                           arsenal cheminParcouru)
```

L'**algorithme** est visualisable en fin de page précédente. Le **raisonnement Algorithmique** est le suivant :

- Tout d'abord, notre fonction reçoit en paramètre une **case** (celle où l'on se trouve), la **carte** (de l'espace d'états, c'est-à-dire map), une **profondeur** (afin de pouvoir arrêter le programme et de respecter l'énoncé), les listes **HorcruxesDescription**, **HorcruxesMap**, **ArmesMap** (comme décrites dans l'énoncé), puis **l'arsenal** (c'est-à-dire la liste des armes collectées), la liste **HorcruxesDetruites** et enfin le **cheminParcouru** (qui sera évidemment actualisé à chaque appel récursif).
- Dans un premier temps, nous ajoutons la **case** passée en paramètre au **cheminParcouru** afin de ne plus pouvoir l'explorer plus tard.
- Dans un second temps, nous étudions la **case** dans laquelle nous nous trouvons.
  - On regarde si une **Méthode de Destruction** s'y trouve.
    - Si c'est le cas, alors nous l'ajoutons à **arsenal** et nous la supprimons de **ArmesMap** afin de ne pas la ramasser plusieurs fois.
  - On regarde si une **Horcruxe** s'y trouve
    - Si c'est le cas, on regarde si nous possédons l'arme adéquate pour la détruire.
      - Si c'est le cas, on ajoute l'Horcruxe à **HorcruxesDetruites**.
    - Dans tous les cas, nous supprimons l'Horcruxe de **HorcruxesMap**. En effet, peu importe qu'elle soit détruite ou non, on ne pourra pas la (re-)détruire en repassant plus tard comme le stipule l'énoncé.
- Une fois cela fait, si la **profondeur** actuelle est **inférieure à 7** (notons que nous pourrions passer en paramètre une variable ProfondeurMax et réaliser un test (profondeur < ProfondeurMax) afin de rendre l'algorithme plus polyvalent), alors nous passons à l'**étude des successeurs** qui n'ont pas encore été visités
  - On initialise d'abord la liste **cases\_suivantes** (variable locale) qui prend comme valeur la liste des successeurs valide de la case en fonction de la **carte** et du **chemin déjà parcouru**.
  - Pour **chaque successeur valide**, nous lançons un **appel récursif** de la fonction **Parcours-Harry** en prenant le soin **d'incrémenter** la valeur de la **profondeur** et en passant les **paramètres actualisés** par le début de l'algorithme.
    - Une fois l'appel récursif réalisé, on **récupère** les données renvoyées par ce dernier (comme nous le verrons à la suivante et dernière étape de l'algorithme) et on **actualise** nos paramètres.
- Une fois tout cela fait, on retourne une liste contenant les éléments susceptibles d'avoir été modifiés (c'est-à-dire **HorcruxesMaps**, **HorcruxesDetruites**, **ArmesMap**, **arsenal** et **cheminParcouru**) afin que, lorsque nous remontons à un noeud père, les modifications soient prises en compte.

## 2 - Code LISP

Pour cette partie, nous ne reparlerons pas vraiment de la partie algorithmique (sauf pour les sous-fonctions que nous allons vous présenter une par une) mais plutôt **des solutions techniques utilisées** pour mettre en pratique notre algorithme en LISP. Avant de démarrer, veuillez noter qu'en raison de la taille de la méthode, nous découperons la fonction en plusieurs parties afin d'expliquer de manière claire et concise. Pour visualiser la fonction dans son entièreté, veuillez vous référer au fichier allegro lié à ce rapport.

Mais avant d'étudier la fonction principale, penchons nous sur les nouvelles sous-fonctions que nous avons écrites pour rendre notre travail plus lisible. La première d'entre elles est la fonction **methodePresente**. Cette dernière reçoit une **case** et la carte des armes **ArmesMap** et renvoie l'**arme présente** dans la case si elle existe (renvoie **nil** sinon).

```
(defun methodePresente (case ArmesMap)
  (cadr (assoc case ArmesMap)))
```

Raisonnement algorithmique : On recherche la ligne correspondant à l'arme recherchée dans la **carte des armes** et on renvoie son **cadr** (c'est-à-dire son nom) si elle existe. A titre d'exemple :

```
CG-USER(12):
(methodePresente '3 armesMap) donne "Crochet de Basilic"
```

```
CG-USER(14):
(methodePresente '28 armesMap) donne NIL
```

La seconde sous-méthode est la méthode **supprimeArmeCarte** qui prend en paramètre une **case** et la **carte des armes** et qui renvoie la **carte avec l'arme** (présente en la case passée en paramètre) **supprimée**.

```
(defun supprimeArmeCarte(case ArmesMap)
  (setq ArmesMap (remove (assoc case ArmesMap) ArmesMap)))
```

Raisonnement algorithmique : On redéfinit **ArmesMap** en **supprimant** la ligne décrivant la méthode de destruction que l'on souhaite **supprimer** (on la retrouve dans la carte grâce à un **assoc**). La carte des armes sans celle supprimée est retournée à la fin de la fonction. A titre d'exemple :

```
CG-USER(16):
((32 "Feudeymon")
 (25 "Epée de Gryffondor")
 (20 "Sortilège de la Mort"))
(supprimeArmeCarte '3 armesMap) renvoie
```

Dans la fonction principale, vous pourrez retrouver également l'utilisation des sous-fonctions **horcruxePresente** et **supprimeHorcruxeCarte**. Cependant, nous ne détaillerons pas leur fonctionnement, ces dernières possédant le même raisonnement que les fonctions ci-dessus (avec une adaptation à la structure de la carte des Horcruxes).

Ainsi, nous pouvons passer à l'étude de la fonction principale.

Tout d'abord notons les paramètres de la fonction qui sont les suivants :

```
(defun Recherche-Harry (case carte profondeur HorcruxesDescriptions cheminParcouru  
HorcruxesMap HorcruxesDetruites ArmesMap ArmesPossedees)
```

On remarque que ceux-ci sont identiques à ceux présentés lors du détail de l'algorithme, cependant, nous pouvons noter que l'ajout d'un argument profondeurMax (qui ne changerait pas avec les appels récursifs) pourrait être utile dans le cas où nous souhaiterions tester le programme sur plusieurs profondeurs différentes.

## 1ère Partie : Affichage de la position et Ajout de la case au chemin

```
(format t "~~% Harry est à la case ~s" case)  
(push case cheminParcouru)
```

Pour l'affichage, on utilise simplement un **format t** en passant **case** comme variable. L'ordre du **cheminParcouru** n'est à priori pas important pour le moment donc on utilise un simple **push** pour ajouter la case au **cheminParcouru** (le chemin sera inversé). Si jamais on souhaite plus tard retrouver le chemin parcouru dans l'ordre, on utilisera simplement la primitive **reverse**.

## 2ème Partie : Arme dans la case ?

```
(setq armePresente (methodePresente case ArmesMap))  
(if armePresente  
  (progn  
    (format t "~~% Arme présente : ~s" armePresente)  
    (push armePresente ArmesPossedees)  
    (setq ArmesMap (supprimeArmeCarte case ArmesMap))  
    (format t "~~% Méthodes de Destruction récupérées :")  
    (dolist (arme ArmesPossedees)  
      (format t "~~% ~s" arme))  
    (format t "~~%~% Passage à la case suivante..."))  
)
```

On commence par initialiser la variable **armePresente** avec le résultat de l'appel de la fonction **methodePresente**. Si une arme est présente dans la case alors **armePresente** contiendra son **nom** (sinon elle contiendra **nil**). Pour suivre, si **armePresente** contient une **autre valeur que nil** (c'est-à-dire si une arme est présente dans la case) alors on réalise les actions qui vont suivre (sinon, on passe à la prochaine partie du programme). D'abord, on **affiche** le nom de l'arme présente dans la case avec un **format t** puis on **l'ajoute** à la liste des armes possédées avec un **push** (l'ordre de l'inventaire n'importe pas). Une fois cela fait, on supprime l'arme de la carte des armes pour ne pas la ramasser une autre fois. Cela se fait grâce à la sous-fonction **supprimeArmeCarte**. Pour finir, on parcourt la liste des armes possédées avec un **dolist** et on affiche chacune des armes avant d'afficher un message signifiant qu'on passe à l'étude des cases suivantes (sachant qu'à ce stade du TP, une horcruxe et une méthode de destruction ne peuvent être cachés à la même case - étant donné que Harry Potter n'est pas encore considéré comme une horcruxe).

### 3ème Partie : Horcruxe dans la case ?

```
(setq HorcruxePresente (horcruxePresente case HorcruxesMap))
(if HorcruxePresente
  (progn
    (format t "~~~ Horcruxes présente : ~s" HorcruxePresente)
    (if (hasBonneArme HorcruxePresente HorcruxesDescriptions ArmesPossedees)
        (progn
          (push HorcruxePresente HorcruxesDetruites)
          (format t "~% Horcruxes détruites :")
          (dolist (horcruxe HorcruxesDetruites)
            (format t "~~~ ~s horcruxe"))
          (setq HorcruxesMap (supprimeHorcruxeCarte case HorcruxesMap))
          (format t "~~~ Passage à la case suivante..."))
        )
      (progn
        (format t "~~% La méthode de destruction nécessaire n'est pas possédée !")
        (format t "~~% L'horcruxe n'a pas été détruite et ne pourra plus l'être !")
        (setq HorcruxesMap (supprimeHorcruxeCarte case HorcruxesMap))
        (format t "~~~ Passage à la case suivante..."))
      )
    )
  )
)
```

On commence par initialiser la variable **HorcruxePresente** avec le résultat de l'appel de la fonction **horcruxePresente**. Si une horcruxe est présente dans la case alors **horcruxePresente** contiendra son **nom** (sinon elle contiendra **nil**). Pour suivre, si **horcruxePresente** contient une **autre valeur que nil** (c'est-à-dire si une horcruxe est présente dans la case) alors on réalise les actions qui vont suivre, sinon on passe à la prochaine partie du programme.

D'abord, on **affiche** le nom de l'horcruxe présente dans la case avec un **format t**.

Ensuite, on vérifie si l'arme nécessaire pour détruire l'horcruxe est contenu dans les armes possédées (grâce à la fonction **hasBonneArme**). Si c'est le cas, alors on ajoute **l'horcruxe à la liste des horcruxes détruites** (avec un **push** - l'ordre importe peu), on **affiche** l'ensemble des horcruxes détruites jusque là avec un **dolist** et un **format t**, puis on **supprime l'horcruxe de la carte des horcruxes** (pour ne pas la détruire à nouveau **ultérieurement**) avant de signaler le passage aux cases suivantes. **Sinon** (si on ne possède pas la méthode de destruction adéquate), alors on affiche un message signalant que **l'horcruxe n'a pas été détruite et qu'elle ne pourra plus l'être** avec un **format t** avant de la **supprimer** de la carte des horcruxes (avec la méthode **supprimeHorcruxeCarte**). Enfin on signale le passage aux cases suivantes avec un affichage format t.

### 4ème Partie : études des successeurs valides

```
(if (< profondeur 7)
  (progn
    (setq cases_suivantes (nreverse(successeurs-valides case carte cheminParcouru)))
    ;(dolist (voisin cases_suivantes)
    ;  (reverse (cons voisin (reverse cheminParcouru))))
    (dolist (voisin cases_suivantes)
      (setq data_parcours (Recherche-Harry voisin carte (+ profondeur 1) HorcruxesDescriptions cheminParcouru
                                                     HorcruxesMap HorcruxesDetruites ArmesMap ArmesPossedees))
      (setq HorcruxesDetruites (car data_parcours))
      (setq HorcruxesMap (cadr data_parcours))
      (setq ArmesMap (caddr data_parcours))
      (setq ArmesPossedees (caddar data_parcours))
      (setq cheminParcouru (cdr(cdddr data_parcours))))))
```

Tout d'abord, pour entrer dans cette partie du code, une pose une **condition** sur la **profondeur** actuelle de la recherche (pour restreindre la recherche, comme demandé par l'énoncé - cela dit, notons que même sans cet élément le programme finirait par s'arrêter après avoir visité toutes les cases accessibles de la carte) qui est ici de **7**. Si l'on voulait rendre le programme **polyvalent** de ce point de vue là, on aurait simplement à remplacer le test (**< profondeur 7**) par un test (**< profondeur profondeurMax**) avec **profondeurMax** une variable passée en paramètre comme expliqué plus tôt.

Si la condition est vérifiée (c'est-à-dire que nous n'avons pas encore atteint la profondeur maximum), alors on commence par initialiser **cases\_suivantes** comme étant **la liste des successeurs valides** de la case en fonction du **chemin déjà parcouru** (appel de la sous-fonction **successeurs-valides** définie plus tôt, cependant, on utilise un **nreverse** pour inverser la liste et pouvoir ainsi parcourir les cases dans l'ordre Haut-Droite-Bas-Gauche).

Ensuite, pour chaque voisin dans **cases\_suivantes** (on itère sur **cases\_suivantes** avec un **dolist**), on réalise un **appel récursif** en passant les paramètres **modifiés** (depuis le début de la fonction) et en prenant le soin **d'incrémenter la profondeur** (la valeur de retour de l'appel sera stockée dans la variable **data\_parcours**). Ensuite, on **affecte les données retournées** par l'appel aux **différents paramètres devant être modifiés** avec des **setq** (**HorcruxesDetruites**, **HorcruxesMap**, **ArmesMap**, **ArmesPossedees**, **cheminParcouru**).

## 5ème Partie : Renvoie des données actualisées

```
(list HorcruxesDetruites HorcruxesMap ArmesMap ArmesPossedees cheminParcouru))
```

Une fois tout cela fait, on **renvoie** une **liste** contenant les paramètres devant être modifiés avec un simple **list** en bout de fonction. A la fin de son exécution, la fonction renverra donc tous les paramètres ayant été modifiés au cours des appels récursifs et des tests sur les cases.

A titre **d'exemple** :

```
(Recherche-Harry '1 map 0 horcruxesDescription nil horcruxesMap nil armesMap nil)
```

### Renvoie

```
(("Coupe de Helga Poufsouffle"
  "Nagini"
  "Journal intime de Tom Jedusor")
 ((15 "Bague de Gaunt")
  (29 "Diadème de Rowena Serdaigle"))
 ((32 "Feudeymon")
  (20 "Sortilège de la Mort"))
 ("Epée de Gryffondor"
  "Crochet de Basilic")
 (36 22 27 26 25 24 13 12 8 8 ...))
```

Pour observer l'affichage total qui se fait lors d'une exécution, merci de bien vouloir le faire directement de votre côté depuis allegro (capture d'écran trop imposante).

# Lord Voldemort part à la recherche des Horcruxes

Nous allons réaliser un **algorithme** puis un **code LISP** pour rajouter des déplacements de Voldemort donnés par l'utilisateur en même temps que ceux d'Harry. Le programme fonctionne de la même manière que le programme précédent: la **profondeur maximale** est toujours de 7 et le programme s'arrête de la même manière que dans la fonction précédente, pareil pour les **armes collectées** et les **horcruxes détruites** au cours du parcours. On ajoute une condition qui arrête le programme lorsque Voldemort et Harry se trouvent sur la même case.

## 1) Algorithme précédent modifié

Fonction RechercheHarry(caseHarry, caseVoldemort, carte, profondeur,  
HorcruxesDescriptions, cheminParcouruHarry, cheminParcouruVoldemort,  
HorcruxesMapHarry, HorcruxesMapVoldemort, HorcruxesDetruitesHarry,  
HorcruxesDetruitesVoldemort, ArmesMap, ArmesPossedeesHarry,  
ArmesPossedeesVoldemort)

Si (caseHarry est égale à caseVoldemort) ET ("Sortilège de la Mort" est membre de  
ArmesPossedeesVoldemort)

    Retourner "Harry Potter a été tué par Voldemort"

Ajouter caseHarry à cheminParcouruHarry  
armePresente ← methodePresente(caseHarry, ArmesMap)

Si armePresente dans caseHarry

    Ajouter armePresente à ArmesPossedeesHarry

    Supprimer l'arme de la carte avec supprimeArmeCarte(caseHarry, ArmesMap)

Ajouter caseVoldemort à cheminParcouruVoldemort

armePresente ← methodePresente(caseVoldemort, ArmesMap)

Si armePresente dans caseVoldemort

    Ajouter armePresente à ArmesPossedeesVoldemort

    Supprimer l'arme de la carte avec supprimeArmeCarte(caseVoldemort, ArmesMap)

HorcruxePresente ← horcruxePresente(caseHarry, HorcruxesMap)

Si HorcruxePresente

    Si (hasBonneArme(HorcruxePresente, HorcruxesDescriptions,

        ArmesPossedeesHarry))

        Ajouter HorcruxePresente à HorcruxesDetruitesHarry

        Supprimer l'Horcruxe de la carte avec supprimeHorcruxeCarte(caseHarry,  
            HorcruxesMapHarry)

Sinon

    Supprimer l'Horcruxe de la carte avec supprimeHorcruxeCarte(caseHarry, HorcruxesMapHarry)

HorcruxePresente ← horcruxePresente(caseVoldemort, HorcruxesMap)

Si HorcruxePresente

    Si (hasBonneArme(HorcruxePresente, HorcruxesDescriptions, ArmesPossedeesVoldemort))

        Ajouter HorcruxePresente à HorcruxesDetruitesVoldemort

        Supprimer l'Horcruxe de la carte avec

        supprimeHorcruxeCarte(caseVoldemort, HorcruxesMapVoldemort)

    Sinon

        Supprimer l'Horcruxe de la carte avec  
        supprimeHorcruxeCarte(caseVoldemort, HorcruxesMapVoldemort)

Si (profondeur < 7)

    cases\_suivantes\_harry ← successeurs-valides(caseHarry, carte, cheminParcouruHarry)

    cases\_suivantes\_voldemort ← Assoc caseVoldemort dans carte

    Pour chaque voisin dans cases\_suivantes\_harry

        Ajouter voisin à cheminParcouruHarry

    Afficher "Cases suivantes possibles pour Voldemort :"

    Pour chaque voisin dans cases\_suivantes\_voldemort

        Afficher voisin

    Afficher "Entrer la case choisie : "

    inputCase ← Lire l'entrée utilisateur

    Pour chaque voisin dans cases\_suivantes\_harry

        data\_parcours ← RechercheHarry(voisin, inputCase, carte, profondeur + 1, HorcruxesDescriptions, cheminParcouruHarry, cheminParcouruVoldemort, HorcruxesMapHarry, HorcruxesMapVoldemort, HorcruxesDetruitesHarry, HorcruxesDetruitesVoldemort, ArmesMap, ArmesPossedeesHarry, ArmesPossedeesVoldemort)

    HorcruxesDetruitesHarry ← car de data\_parcours

    HorcruxesDetruitesVoldemort ← cadr de data\_parcours

    HorcruxesMapHarry ← caddr de data\_parcours

    HorcruxesMapVoldemort ← cadddr de data\_parcours

    ArmesMap ← cadr de cddd de data\_parcours

    ArmesPossedeesHarry ← caddr de cddd de data\_parcours

    ArmesPossedeesVoldemort ← cadddr de cddd de data\_parcours

    cheminParcouruHarry ← cadr de cddd de cddd de data\_parcours

    cheminParcouruVoldemort ← caddr de cddd de cddd de data\_parcours

Retourner une liste contenant HorcruxesDetruitesHarry, HorcruxesDetruitesVoldemort, HorcruxesMapHarry, HorcruxesMapVoldemort, ArmesMap, ArmesPossedeesHarry, ArmesPossedeesVoldemort, cheminParcouruHarry, cheminParcouruVoldemort

Fin

## 2) Code LISP

On initialise d'abord une nouvelle liste HorcruxesMapVoldemort et on renomme HorcruxesMap en HorcruxesMapHarry. En effet, les deux personnages auront des maps différentes en fonction des cases qu'ils visiteront.

### Partie 1 : Initialisation et Conditions de Fin

La fonction commence par vérifier si Harry et Voldemort sont sur la même case (equal caseHarry caseVoldemort) et si Voldemort possède le "Sortilège de la Mort" ((member "Sortilège de la Mort" ArmesPossedeesVoldemort :test #'string=)). Si ces deux conditions sont remplies, le programme se termine immédiatement, renvoyant une liste indiquant que Harry a été tué par Voldemort.

```
(defun Recherche-Harry (caseHarry caseVoldemort carte profondeur HorcruxesDescriptions cheminParcouruHarry
                                         cheminParcouruVoldemort HorcruxesMapHarry HorcruxesMapVoldemort HorcruxesDetruitesHarry
                                         HorcruxesDetruitesVoldemort ArmesMap ArmesPossedeesHarry
                                         ArmesPossedeesVoldemort)

  (when (and (equal caseHarry caseVoldemort)
                    (member "Sortilège de la Mort" ArmesPossedeesVoldemort :test #'string=))
    (return-from Recherche-Harry (list "Harry Potter a été tué par Voldemort"))
  )
```

### Partie 2 et 3 : Actions pour Harry et Voldemort

- **Harry :**
  - Affiche la position actuelle de Harry.
  - Ajoute la case à son chemin parcouru.
  - Vérifie la présence d'une arme dans la case et effectue des actions en conséquence.
  - Vérifie la présence d'un horcruxe dans la case et effectue des actions en conséquence.
- **Voldemort :**
  - Affiche la position actuelle de Voldemort.
  - Ajoute la case à son chemin parcouru.
  - Vérifie la présence d'une arme dans la case et effectue des actions en conséquence.
  - Vérifie la présence d'un horcruxe dans la case et effectue des actions en conséquence.

```

'(format t "~~~ Harry est à la case ~s" caseHarry)
(push caseHarry cheminParcouruHarry)
(setq armePresente (methodePresente caseHarry ArmesMap))
(if armePresente
    (progn
        (format t "~~~ Arme présente : ~s" armePresente)
        (push armePresente ArmesPossedeesHarry)
        (setq ArmesMap (supprimeArmeCarte caseHarry ArmesMap))
        (format t "~~~ Méthodes de Destruction récupérées :")
        (dolist (arme ArmesPossedeesHarry)
            (format t "~~~ ~s" arme))
        (format t "~~~ Passage à la case suivante..."))
    )
)

(format t "~~~ Voldemort est à la case ~s" caseVoldemort)
(push caseVoldemort cheminParcouruVoldemort)
(setq armePresente (methodePresente caseVoldemort ArmesMap))
(if armePresente
    (progn
        (format t "~~~ Arme présente : ~s" armePresente)
        (push armePresente ArmesPossedeesVoldemort)
        (setq ArmesMap (supprimeArmeCarte caseVoldemort ArmesMap))
        (format t "~~~ Méthodes de Destruction récupérées :")
        (dolist (arme ArmesPossedeesVoldemort)
            (format t "~~~ ~s" arme))
        (format t "~~~ Passage à la case suivante..."))
    )
)

(setq HorcruxePresente (horcruxePresente caseHarry HorcruxesMapHarry))
(if HorcruxePresente
    (progn
        (format t "~~~ Horcruxes présente sur la case Harry: ~s" HorcruxePresente)
        (if (hasBonneArme HorcruxePresente HorcruxesDescriptions ArmesPossedeesHarry)
            (progn
                (push HorcruxePresente HorcruxesDetruitHarry)
                (format t "~~~ Horcruxes détruites par Harry :")
                (dolist (horcruxe HorcruxesDetruitHarry)
                    (format t "~~~ ~s" horcruxe))
                (setq HorcruxesMapHarry (supprimeHorcruxecarte caseHarry HorcruxesMapHarry))
                (format t "~~~ Passage à la case suivante..."))
            )
        (progn
            (format t "~~~ La méthode de destruction nécessaire n'est pas possédée !")
            (format t "~~~ L'Horcruxe n'a pas été détruite et ne pourra plus l'être !")
            (setq HorcruxesMapHarry (supprimeHorcruxecarte caseHarry HorcruxesMapHarry))
            (format t "~~~ Passage à la case suivante..."))
        )
    )
)
)
)

```

```

(setq HorcruxePresente (horcruxePresente caseVoldemort HorcruxesMapVoldemort))
(if HorcruxePresente
  (progn
    (format t "~~~ Horcruxes présente : ~s" HorcruxePresente)
    (if (hasBonneArme HorcruxePresente HorcruxesDescriptions ArmesPossedeesVoldemort)
        (progn
          (push HorcruxePresente HorcruxesDetruitesVoldemort)
          (format t "~~~ Horcruxes détruites :")
          (dolist (horcruxe HorcruxesDetruitesVoldemort)
            (format t "~~~ ~s" horcruxe))
          (setq HorcruxesMapVoldemort (supprimeHorcruxeCarte caseVoldemort HorcruxesMapVoldemort))
          (format t "~~~ Passage à la case suivante..."))
        )
      (progn
        (format t "~~~ La méthode de destruction nécessaire n'est pas possédée !")
        (format t "~~~ L'Horcruxe n'a pas été détruite et ne pourra plus l'être !")
        (setq HorcruxesMapVoldemort (supprimeHorcruxeCarte caseVoldemort HorcruxesMapVoldemort))
        (format t "~~~ Passage à la case suivante..."))
      )
    )
  )
)

```

## Partie 4 : Exploration des Successeurs

- Vérifie si la profondeur actuelle est inférieure à 7 (ou **profondeurMax** si cela est fourni en tant qu'argument).
- Si la condition est vérifiée, récupère les cases voisines valides pour Harry.
- Affiche les cases possibles pour Voldemort.
- Demande à l'utilisateur de choisir une case pour Voldemort (**inputCase**).
- Effectue une recherche récursive pour chaque case voisine de Harry avec les paramètres mis à jour.

```

(if (< profondeur 7)
  (progn
    (setq cases_suivantes_harry (successeurs_valides caseHarry carte cheminParcouruHarry))
    (setq cases_suivantes_voldemort (assoc caseVoldemort carte))
    (dolist (voisin cases_suivantes_harry)
      (reverse (cons voisin (reverse cheminParcouruHarry))))
    (format t "~~~ Cases suivantes possibles pour Voldemort :")
    (dolist (voisin cases_suivantes_voldemort)
      (format t "~~~ ~s" voisin))
    (format t "~~~ Entrer la case choisie: ")
    (setq inputCase (read))

    (dolist (voisin cases_suivantes_harry)
      (setq data_parcours (Recherche_Harry voisin inputCase carte (+ profondeur 1) HorcruxesDescriptions cheminParcouruHarry
                                         cheminParcouruVoldemort HorcruxesMapHarry HorcruxesMapVoldemort Horcruxes
                                         HorcruxesDetruitesVoldemort ArmesMap ArmesPossedeesHarry
                                         ArmesPossedeesVoldemort))
      (setq HorcruxesDetruitesHarry (car data_parcours))
      (setq HorcruxesDetruitesVoldemort (cdr data_parcours))
      (setq HorcruxesMapHarry (caddr data_parcours))
      (setq HorcruxesMapVoldemort (caddr data_parcours))
      (setq ArmesMap (cadr (cdddr data_parcours)))
      (setq ArmesPossedeesHarry (caddr (cdddr data_parcours)))
      (setq ArmesPossedeesVoldemort (caddr (cdddr (cdddr data_parcours))))
      (setq cheminParcouruHarry (cadr (cdddr (cdddr data_parcours))))
      (setq cheminParcouruVoldemort (caddr (cdddr (cdddr data_parcours)))))
      (setq cheminParcouruVoldemort (caddr (cdddr (cdddr (cdddr data_parcours)))))))
    (list HorcruxesDetruitesHarry HorcruxesDetruitesVoldemort HorcruxesMapHarry HorcruxesMapVoldemort ArmesMap ArmesPossedeesHarry)
  )
)

```

## Partie 5 : Renvoi des Données Actualisées

- Renvoie une liste contenant les paramètres mis à jour après l'exploration des cases.
- Les données renvoyées incluent les horcruxes détruites par Harry et Voldemort, les cartes des horcruxes et des armes mises à jour, les armes possédées, et les chemins parcourus.

L'exécution de la fonction s'affiche de cette manière à chaque fois qu'Harry explore une nouvelle case :

CG-USÉR(104):

Harry est à la case 1

Voldemort est à la case 20

Arme présente : "Sortilège de la Mort"

Méthodes de Destruction récupérées :

"Sortilège de la Mort"

Passage à la case suivante...

Cases suivantes possibles pour Voldemort :

20

21

29

Entrer la case choisie:

---

# Créativité

Voici des améliorations créatives pour enrichir l'expérience de jeu.

## 1. Représentation visuelle améliorée en ASCII

Pour offrir une expérience visuelle plus immersive, une amélioration consiste à proposer une représentation graphique en ASCII du labyrinthe à chaque avancement dans un nouveau nœud du jeu. Cette approche simple, mais efficace, utilise la notation ASCII pour représenter chaque case du labyrinthe. Les cases vides explorables sont symbolisées par [ ], celles qui ne le sont pas par [X], les cases occupées par Harry et Voldemort par [H] et [V] respectivement.

De plus, cette représentation pourrait également inclure des indications visuelles sur la présence d'armes et d'Horcruxes dans le labyrinthe. Cela permettrait à l'utilisateur de planifier ses mouvements en fonction des emplacements des armes et horcruxes.

Ainsi, il suffirait de construire une fonction qui génère à chaque tour le labyrinthe. En vérifiant la caseHarry et caseVoldemort, la fonction généreraient les positions respectives par la notation donnée et le reste des cases par [ ] ou [X].

Exemple d'algorithme:

```
genererLabyrinthe(labyrinthe, caseHarry, caseVoldemort, armes, horcruxes):
    pour chaque ligne du labyrinthe:
        pour chaque colonne du labyrinthe:
            si la case est la position de Harry:
                afficher [H]
            sinon si la case est la position de Voldemort:
                afficher [V]
            sinon si la case contient une arme:
                afficher [A]
            sinon si la case contient un Horcruxe:
                afficher [h]
            sinon:
                Si explorable :
                    afficher [ ]
                Sinon :
                    afficher [X]
            afficher nouvelle ligne
```

## 2. Mise en place d'éénigmes pour ramasser des armes

Cette proposition vise à intégrer des défis intellectuels, renforçant l'aspect stratégique du jeu pour offrir une expérience plus interactive pour l'utilisateur.

Des énigmes variées et adaptées à l'univers d'Harry Potter seraient préalablement définies pour chaque arme. L'utilisateur, Voldemort, devrait résoudre l'énigme pour récupérer l'arme.

Cela nécessiterait une connaissance de l'univers d'Harry Potter. En cas d'échec, l'arme ne serait pas récupérée et l'utilisateur aurait la possibilité de rester sur la case pour retenir alors qu'Harry continue son exploration.

Un exemple d'algorithme:

```
resoudreEnigme(arme, enigme):
    afficher enigme
    reponse = saisieUtilisateur("Entrez votre réponse: ")
    si reponse est correcte:
        retourner True
    sinon:
        retourner False
```

Il faudrait alors l'implémenter dans la fonction principale dans la partie où on récupère les armes, il faudrait ajouter la condition suivante:

```
si resoudreEnigme(arme, enigme):
    ajouter arme à armesVoldemort
```

### 3. Mode multijoueur

#### Mode Multijoueur Stratégique : Conflit Entre Sorciers

Cette amélioration consiste à introduire un mode multijoueur où les joueurs peuvent incarner Lord Voldemort comme précédemment mais aussi Harry Potter. Chaque joueur contrôle activement son personnage, prenant des décisions stratégiques pour trouver et détruire les Horcruxes ou, dans le cas de Voldemort, éliminer Harry.

Ainsi, il suffirait de proposer un choix du mode au début du programme. Si le mode choisi est le mode classique, on effectue le programme normalement avec la recherche en profondeur pour Harry et les entrées utilisateur pour Voldemort. Sinon, on applique le code fait pour les mouvements de Voldemort pour Harry aussi, en ignorant la recherche en profondeur. Il faudrait donc une condition au début du programme qu'on passera en paramètre et qui régit la suite du fonctionnement pour le reste de son exécution.

Il faudrait s'assurer que les rôles sont équilibrés avec l'emplacement des différentes armes et horcruxes sur la map pour que le même personnage ne gagne pas à chaque tour.

# Conclusion

Tout d'abord, nous pouvons dire que ce TP nous aura permis de grandement consolider nos connaissances en recherche d'états à travers l'implémentation du programme LISP basé sur l'univers d'Harry Potter. L'implémentation des algorithmes de recherche en profondeur est aussi une partie importante et cruciale de ce TP.

L'analyse des arbres de recherche en profondeur au départ des cases 1 et 36 nous a permis de comprendre les limitations de la recherche à partir de certaines positions, notamment la difficulté à détruire tous les Horcruxes en partant de la case 1 ou de la case 36, même en ne posant aucune limite à la profondeur.

La mise en place de fonctions de service telles que la recherche des successeurs valides, la détection des armes présentes dans une case, et la vérification des conditions pour la destruction des Horcruxes ont été cruciales pour la réalisation de l'algorithme de recherche en profondeur.

De plus, l'extension du programme pour inclure les déplacements de Lord Voldemort a ajouté une dimension interactive intéressante. Les actions simultanées d'Harry et de Voldemort, avec la possibilité de s'affronter si les conditions sont réunies, ont renforcé la complexité et la richesse de l'expérience de jeu.

Ce TP nous a permis d'approfondir encore nos compétences en programmation et en résolution de problèmes algorithmiques tout en explorant le contexte ludique basé sur l'univers d'Harry Potter introduit durant le médian.