Faculté sciences & sciences de l'ingénieur
Université Bretagne Sud

Projet : Agent logique pour la chasse au Wumpus

Travail présenté à Salim Lardjane
INF1604

réalisé par
Jean Sevaux, Aude Le Boutouiller,
Rafael Massampu Evora Tavares

25 avril 2021

Table des matières

1 Introduction

Le jeu «`Hunt the Wumpus`», est sorti initialement en 1972. Le but de ce dernier est de guider le chasseur vers le trésor puis vers la case départ, tout en faisant attention aux obstacles, dont le Wumpus, mais aussi des puits. Il est possible d'éliminer le monstre, même s'il est préférable de l'esquiver, par contre, les puits ne peuvent être détruits ou bouchés, ils seront obligatoirement contournés par le chasseur.

2 Rappel de la problématique

Dans le cadre de l'enseignement de Logique en ce sixième semestre de Licence, il nous est proposé un projet, qui consiste à programmer le jeu «`Hunt the Wumpus`», permettant de mettre en pratique nos connaissances sur la logique des propositions (LP0).

Le but du projet est de construire la base de connaissance de l'agent ainsi que l'implémentation de ses routines de recherches. Il s'agit donc de la réalisation d'une intelligence artificielle, programmée en Python, qui créer un agent capable de résoudre le jeu de «`La chasse au Wumpus`».

Avec cela, le but du projet n'est pas de coder seulement le jeu mais aussi un programme, une intelligence artificielle donc, capable de résoudre toute seule le jeu. Le jeu doit être codé en Python, avec une carte du jeu pour qu'on soit capable de se repérer pendant le jeu et que le programme n'ait aucune aide de l'humain pour résoudre. Donc avec tout cela en tête nous avons apporté la solution décrite ci-dessous.

3 Mise en place et configuration du projet

- CryptoMiniSat

  Parmi les membres de cette équipe, deux possèdent des ordinateurs sous Windows 10, le 3ème un ordinateur sous Debian. Bien que le solveur SAT CryptoMiniSat peut être compilé sous Windows, nous avons tous décidé de l'installer sous une configuration Unix/Linux. D'une part pour faciliter les échanges et le travail d'équipe en partant tous de la même base, d'autre part puisque nous avons l'habitude de programmer et de compiler sous cette distribution.

  Cela nous a été possible, grâce à un sous-système Ubuntu pour Windows (WSL). Avec nos configurations Linux opérationnelles, nous n'avions plus qu'à suivre les instructions de compilation disponibles dans le README présent sur le GitHub du projet CryptoMiniSat.

- Le jeu et les tests

La deuxième partie de la mise en place portait sur l'installation du projet. Pour cela, nous avons téléchargé l'archive disponible sur le site web de présentation du projet.

Afin de vérifier que notre projet était opérationnel, c'est-à-dire que le solveur CryptoMiniSat et le jeu avaient bien été installés et configurés, une série de tests y est intégrée.

Le lancement de ces tests se fait depuis le dossier racine du projet en lançant la commande `python3.9 wumpus.py -t`.

L'installation fut correcte dès que nous avons obtenu ce résultat :

```
Running simple CryptoMiniSat test:
-------------------------------------------------------
Test 1
  Query:      '(P | ~P)'
  Query CNF:  [(P | ~P)]
  Result:     True   (Expected: True)
  Variable Assignment: {P: False}
-------------------------------------------------------
Test 2
  Query:      '(P & ~P)'
  Query CNF:  [P, ~P]
  Result:     False   (Expected: False)
-------------------------------------------------------
Test 3
  Query:      '(P | R) <=> (~(Q | R) & (R >> ~(S <=> T)))'
  Query CNF:  [(S | ~T | P | R | Q | R), (T | ~S | P | R | Q | R), (R | P | R | Q | R), (~P | ~Q), (~R | ~Q), (~P | ~R), (~R | ~R), (~P | ~T | ~S | ~R),
(~R | ~T | ~S | ~R), (~P | S | ~S | ~R), (~R | S | ~S | ~R), (~P | ~T | T | ~R), (~R | ~T | T | ~R), (~P | S | T | ~R), (~R | S | T | ~R)]
  Result:     True   (Expected: True)
  Variable Assignment: {P: False, Q: True, R: False, S: False, T: False}
-------------------------------------------------------
Successfully passed 3 tests.
DONE.
```

L'avant dernière phrase nous indique que les 3 tests ont été réalisés avec succès, nous avons donc bien installé le tout, nous pouvons maintenant commencer à implémenter les fonctions de ce projet.

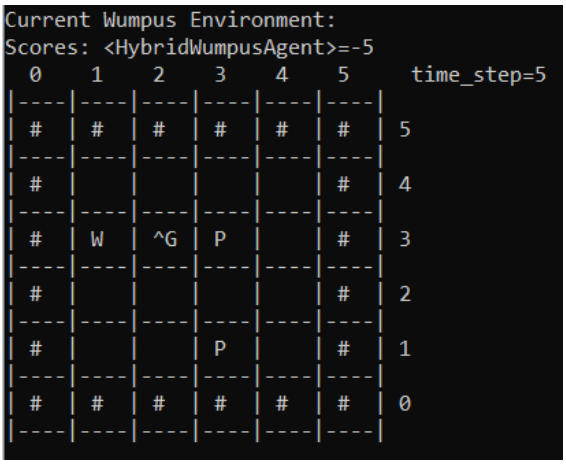
Un autre test est aussi possible, il est accessible par la commande `python3.9 wumpus.py -k`. Il permet à l'utilisateur d'avoir la main sur les mouvements du chasseur et de choisir sa stratégie ainsi que ses directions. Cette commande nous a également permis d'appréhender ce projet, comprendre son fonctionnement, les stratégies à adopter.

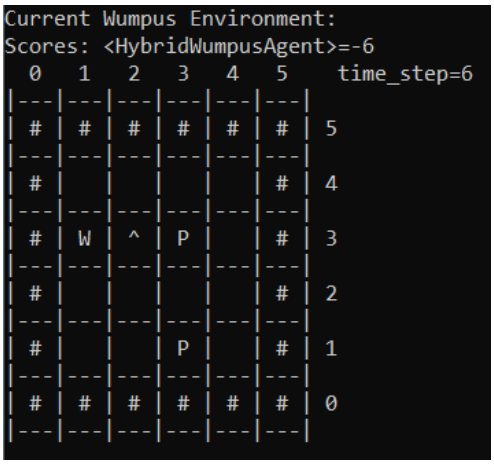
À chaque déplacement, le jeu nous indique les perceptions du chasseur, c'est alors à nous de décider de la direction qu'il doit prendre, toujours dans le but de récupérer le trésor sans croiser le chemin du Wumpus.

```
[0] You perceive: ['~Stench', '~Breeze', '~Glitter', '~Bump', '~Scream']
Enter Action ('?' for list of commands): ?
```

```
   The following are valid Hunt The Wumpus actions:
     'TurnRight', 'TurnLeft', 'Forward', 'Grab', 'Climb', 'Shoot', 'Wait'
```

Un fois le chasseur arrivé sur la case du trésor (image de gauche), il faut qu'il le saisisse avec l'instruction `"GRAB"` (image de droite) :





Mais le jeu ne s'arrête pas là, il faut que le chasseur revienne à son point de départ afin de remonter avec son trésor, sans oublier le Wumpus qu'il faut éviter, soit en le contournant, soit en le tuant. Une fois que le chasseur a réalisé l'instruction `"Climb"` depuis la case de départ, il a gagné contre le Wumpus, le jeu s'arrête et son score s'affiche :

```
Final Scores: <HybridWumpusAgent>=978
```

Étant donné que les déplacements, le tir de flèche pour tuer le Wumpus, le fait de tomber dans un puits ou de se faire manger par le monstre coûte des points, et que la victoire du chasseur en partant avec le trésor rapporte des points, le but de ce jeu est d'obtenir le plus grand nombre de points.

4 Développement du projet

Lorsque les fonctions sont bien implémentées, il nous est possible de tester la version "automatisée" de cette chasse grâce à la commande `python3.9 wumpus.py -k`. Voici les résultats que nous obtenons avec notre code :

```
Scores: <HybridWumpusAgent>=0
  0   1   2   3   4   5    time_step=0
|---|---|---|---|---|---|
| # | # | # | # | # | # | 5
|---|---|---|---|---|---|
| # |   |   |   |   | # | 4
|---|---|---|---|---|---|
| # | W | G | P |   | # | 3
|---|---|---|---|---|---|
| # |   |   |   |   | # | 2
|---|---|---|---|---|---|
| # | ^ |   | P |   | # | 1
|---|---|---|---|---|---|
| # | # | # | # | # | # | 0
|---|---|---|---|---|---|
```

```
Ask if Wumpus is Alive:
  Is Wumpus Alive? : True
```

Nous pouvons apercevoir qu'au lancement du jeu, notre chasseur se situe toujours dans la case (1,1), il est représenté par le symbole "^", et que le Wumpus est bien en vie.

```
<HybridWumpusAgent> perceives ['Stench', '~Breeze', '~Glitter', '~Bump', '~Scream']
```

Étant donné qu'à partir de cette case, les indicateurs de perceptions sont tous précédés d'un ' ', il n'y a donc aucun danger dans les cases voisines, le chasseur étend alors sa connaissance du plateau de jeu et peut sauvegarder ces informations de la manière suivante :

```
Find OK locations queries
  0             1             2             3             4             5             time_step=0

|-------------|-------------|-------------|-------------|-------------|-------------|
| #           | #           | #           | #           | #           | #           | 5
|-------------|-------------|-------------|-------------|-------------|-------------|
| #           | OK1_4_0=?   | OK2_4_0=?   | OK3_4_0=?   | OK4_4_0=?   | #           | 4
|-------------|-------------|-------------|-------------|-------------|-------------|
| #           | OK1_3_0=?   | OK2_3_0=?   | OK3_3_0=?   | OK4_3_0=?   | #           | 3
|-------------|-------------|-------------|-------------|-------------|-------------|
| #           | OK1_2_0=True| OK2_2_0=?   | OK3_2_0=?   | OK4_2_0=?   | #           | 2
|-------------|-------------|-------------|-------------|-------------|-------------|
| #           | OK1_1_0=True| OK2_1_0=True| OK3_1_0=?   | OK4_1_0=?   | #           | 1
|-------------|-------------|-------------|-------------|-------------|-------------|
| #           | #           | #           | #           | #           | #           | 0
|-------------|-------------|-------------|-------------|-------------|-------------|
```

Dans ce tableau, le jeu enregistre les cases qui sont sans danger grâce à la valeur "TRUE" qui leur est attribuée. C'est lors de l'initialisation des cases à "TRUE" que la logique intervient. En effet, si le chasseur est sur une case sans aucune perception, cela signifie qu'aucune des cases voisines ne comportent un danger, elles peuvent donc être explorées librement.

Un autre tableau sert à évoluer dans le jeu, celui-ci contient également les cases qui n'ont pas de danger mais qui n'ont pas été visitées non plus. Cette base de connaissance est utile au chasseur pour qu'il soit efficace dans sa recherche.

En effet, chaque déplacement coûte 1 point, et l'objectif étant de finir le jeu avec le plus

de points possibles, il est important de les optimiser et d'aller explorer des cases nouvelles qui nous apporteront de nouvelles connaissances.

```
Safe univisited locations:
  0          1          2          3          4          5           time_step=0
|----------|----------|----------|----------|----------|----------|
| #        | #        | #        | #        | #        | #        | 5
|----------|----------|----------|----------|----------|----------|
| #        |          |          |          |          | #        | 4
|----------|----------|----------|----------|----------|----------|
| #        |          |          |          |          | #        | 3
|----------|----------|----------|----------|----------|----------|
| #        | L1_2_0=T |          |          |          | #        | 2
|----------|----------|----------|----------|----------|----------|
| #        |          | L2_1_0=T |          |          | #        | 1
|----------|----------|----------|----------|----------|----------|
| #        | #        | #        | #        | #        | #        | 0
|----------|----------|----------|----------|----------|----------|
```

À partir de cette case et de ses connaissances, le chasseur décide d'aller tout droit :

```
<HybridWumpusAgent> perceives ['~Stench', '~Breeze', '~Glitter', '~Bump', '~Scream'] and does Forward
```

Nous le retrouvons donc dans la case supérieure, puisqu'il était dirigé vers le Nord.

```
Current Wumpus Environment:
Scores: <HybridWumpusAgent>=-1
  0   1   2   3   4   5    time_step=1
|---|---|---|---|---|---|
| # | # | # | # | # | # | 5
|---|---|---|---|---|---|
| # |   |   |   |   | # | 4
|---|---|---|---|---|---|
| # | W | G | P |   | # | 3
|---|---|---|---|---|---|
| # | ^ |   |   |   | # | 2
|---|---|---|---|---|---|
| # |   |   | P |   | # | 1
|---|---|---|---|---|---|
| # | # | # | # | # | # | 0
|---|---|---|---|---|---|
```

Depuis cette case, le chasseur détecte une odeur nauséabonde, représentée dans le jeu par la valeur `"Stench"` non précédée d'un `"~"` :

```
<HybridWumpusAgent> perceives ['Stench', '~Breeze', '~Glitter', '~Bump', '~Scream']
```

Il y a donc un danger dans au moins une des cases voisines, mais avec les éléments en sa possession le chasseur n'est pas en mesure de déduire la ou les cases concernées. Dans notre situation, le chasseur décide d'aller tout droit.

```
<HybridWumpusAgent> perceives ['Stench', '~Breeze', '~Glitter', '~Bump', '~Scream'] and does Forward
```

Malheureusement, il atterrit sur la case où se trouve le Wumpus, qui le mange et qui lui fait perdre 1000 points ainsi que la partie.

```
Current Wumpus Environment:
Scores: <HybridWumpusAgent>=-1002
  0    1    2    3    4    5     time_step=2
|----|----|----|----|----|----|
| #  | #  | #  | #  | #  | #  | 5
|----|----|----|----|----|----|
| #  |    |    |    |    | #  | 4
|----|----|----|----|----|----|
| #  | ^W | G  | P  |    | #  | 3
|----|----|----|----|----|----|
| #  |    |    |    |    | #  | 2
|----|----|----|----|----|----|
| #  |    |    | P  |    | #  | 1
|----|----|----|----|----|----|
| #  | #  | #  | #  | #  | #  | 0
|----|----|----|----|----|----|
```

```
A Wumpus ate <HybridWumpusAgent>!
```

Nous obtenons donc notre score final, correspondant à 2 points de perdus pour les déplacements ainsi que 1000 points pour l'arrivée sur la case du Wumpus et la mort du chasseur.

```
Final Scores: <HybridWumpusAgent>=-1002
```

Nous avons une autre version de ce programme, le chasseur ne sort pas vainqueur ici non plus mais nous pouvons analyser son approche différente. Ces programmes diffèrent au niveau de la fonction `axiom_generator_breeze_percept_and_location_property(x, y, t)`.

Comme avec le programme précédent, le chasseur se trouve dans la case (1,1) du plateau de jeu.

```
Scores: <HybridWumpusAgent>=0
  0   1   2   3   4   5    time_step=0
|---|---|---|---|---|---|
| # | # | # | # | # | # | 5
|---|---|---|---|---|---|
| # |   |   |   |   | # | 4
|---|---|---|---|---|---|
| # | W | G | P |   | # | 3
|---|---|---|---|---|---|
| # |   |   |   |   | # | 2
|---|---|---|---|---|---|
| # | ^ |   | P |   | # | 1
|---|---|---|---|---|---|
| # | # | # | # | # | # | 0
|---|---|---|---|---|---|
```

Contrairement à la version précédente, la chasseur décide de tirer immédiatement, cela lui

coûte 10 points. Le Wumpus étant dans cette direction, il est tué, le chasseur en est informé par le cri que pousse le monstre.

```
<HybridWumpusAgent> perceives ['~Stench', '~Breeze', '~Glitter', '~Bump', 'Scream']
```

Dans la suite du jeu, cette information est conservée et on la retrouve à chaque étape.

```
Ask if Wumpus is Alive:
  Is Wumpus Alive? : False
```

Par la suite, notre chasseur décide d'avancer tout droit à 2 reprises, il arrive donc sur la case où se trouvait initialement le Wumpus, étant donné qu'il a été tué, sa mort étant représentée par le "X", le chasseur peut passer librement sur cette case.

```
Current Wumpus Environment:
Scores: <HybridWumpusAgent>=-14
  0    1    2    3    4    5     time_step=4
|----|----|----|----|----|----|
| #  | #  | #  | #  | #  | #  | 5
|----|----|----|----|----|----|
| #  |    |    |    |    | #  | 4
|----|----|----|----|----|----|
| #  | ^X | G  | P  |    | #  | 3
|----|----|----|----|----|----|
| #  |    |    |    |    | #  | 2
|----|----|----|----|----|----|
| #  |    |    | P  |    | #  | 1
|----|----|----|----|----|----|
| #  | #  | #  | #  | #  | #  | 0
|----|----|----|----|----|----|
```

Malheureusement, le chasseur continue d'aller tout droit indéfiniment, il ne trouve donc jamais le trésor et ne gagne pas la partie.

5 Conclusion

En conclusion, nous pouvons dire, que malgré le temps qui nous était compté, et le nombre de corrections à apporter au programme, nous avons essayé de tout mettre en œuvre pour pouvoir réussir, et ainsi obtenir des résultats encourageants.

En effet, nonobstant un bon départ, nous n'avons malheureusement pas réussi à aller au bout de cette chasse au trésor. Cependant, comme nous avons pu le constater précédemment, nos deux versions proposent des résultats satisfaisants.

Bien que très intéressante, l'implémentation d'une intelligence artificielle reste complexe, les décisions et déductions qui nous paraissent évidentes, sont en réalité plus complexes à programmer.

De plus, ce sont des programmes qui peuvent constamment être améliorés. C'est pourquoi notre réflexion et nos tests sur ce projet continuent, dans le but de rendre notre programme, et donc le chasseur, de plus en plus intelligent.