# TP1

**Exercice 1 :**

**.1)**

Voici comment sont organisées les salles :

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

**porte(2,1)** → veut dire qu'il existe une porte qui va de la salle 2 à la salle 1.

Chaque porte est défini comme ça au début du fichier .pl.

**.2)** Dans la salle (1) (haut à gauche), il existe une sortie, qui est défini comme **sortie(1)** dans le fichier .pl, toute sortie, entree et minotaure seront défini ainsi.

**Exercice 2 :**

**.1)** Règle **chemin(De,Vers)** return true si le chemin depuis De à Vers existe, false sinon.

**chemin(X,X).** règle de base, qui dit qu'il existe un chemin de X à X **(De = Vers)**.

Si **De != Vers**, alors on "demande" à prolog s'il existe une porte sortante de **De**, avec la commande **porte(De, X)**, cette commande retourne dans **X** toute salle accessible depuis **De**. Ensuite nous appelons récursivement en remplacent **De** par le **X** trouvé, la logique, est de plutôt que calculer/chercher, un chemin de **De** à **Vers**, on cherche s'il existe une porte de **De** à **X**, et ensuite un chemin de **X** à **Vers**.

**.2)** Règle **itineraire(De,Vers,Pieces)**, retourne dans **Pieces** une liste des salles à parcourir pour aller de **De** à **Vers** (si le chemin existe).

La logique appliqué dans cette règle est la même que dans **chemin(De,Vers)**.

**chemin(X,X,[X]).** règle de base, quand **De** = **Vers**, alors on retourne sous forme de liste, l'élément **De**.

On initialise le backtracking pour construire la liste, vu que la valeur **Vers** est déjà dans la liste, du à la règle de base. Du coup j'ai fait un **(if -> then ; else)** qui vérifie qu'on est pas dans l'étape ou **X** = **Vers**, si :

**X !**= **Vers** → on concatène la liste précédente avec **[X]**.

**X** = **Vers** alors on concatène une liste vide à la liste précédente, car cette liste précédente est exactement **[X]**.

**! ! !**

Par exemple, pour **itineraire(3,1,Pieces)** → **Pieces** = **[3, 2,1]**, veut dire qu'en partant de la salle 3, on prend la porte vers la salle 2, et ensuite pour la salle 1.

**! ! !**

**Exercice 3 :**

**.1)** La règle **batterie(Pieces,Batterie,Reste)** retourne dans **Reste** la batterie restante après avoir parcourue les salles dans **Pieces**. Cette règle est très simple, elle calcule la taille de la liste **Pieces** et soustrait cette valeur à **Batterie**. Par contre, si il n'y a pas assez de batterie pour parcourir toutes les salles, la règle retourne **false**.

Mon code retourne un résultat attendu lors que je tente la règle **test_batterie(De,Vers,Batterie,Reste)**.

**.2)** La règle **chemin_batterie(De,Vers,Batterie,Pieces,Reste)** retourne dans **Pieces** un chemin qui va de **De** à **Vers**, elle utilise donc la règle **itineraire(De,Vers,Pieces)** définie avant dans le TP, ensuite avec la liste **Pieces** elle utilise la règle **batterie(Pieces,Batterie,Reste)** pour savoir si ce chemin **Pieces** est réalisable avec la **Batterie** à disposition. La règle **batterie(Pieces,Batterie,Reste)** retourne déjà **false** si le chemin n'est pas réalisable, donc je n'ai pas besoin de faire d'autres changements.

**.3)** La règle **chemin_reussite(Batterie,Pieces)**, cherche les sorties, entrees et le minotaure avec les commandes **entree(X), sortie(Y) et minotaure(Z)** respectivement, ensite on utilise la règle **chemin_batterie(X,Y,Batterie,Pieces,Reste)**, qui nous rend tous les chemins possibles de **X** à **Y** avec la **Batterie** disponible, ensuite avec la règle prédéfinie **member(Z,Pieces)**, on trie les chemins qui contiennent l'enplacement du Minotaure(**Z**).

**Exercice 4 :**

**.1.1)** La règle **reussite_complete(Batterie,Pieces)**, cherche les sorties, entrees et le minotaure avec les commandes **entree(X), sortie(Y) et minotaure(Z)** respectivement, ensuite un appel à la règle **itineraire(X,Y,Pieces)** qui retourne un chemin dans **Pieces** qui va de **X** à **Y**, ensuite on vérifie si dans ce chemin il y a la salle **Z** (minotaure) :

Si **oui** → on utilise la règle **batterie(Pieces,Batterie,Reste)**, sauf que ici la valeur dans **Batterie** vaut **Batterie = Batterie - 5 (énergie lumière) - 2 (énergie tweet)**.

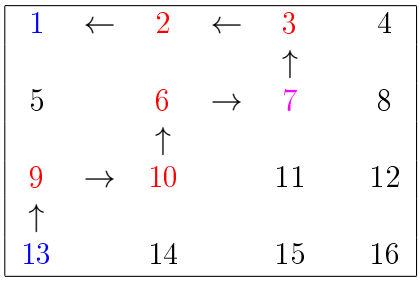
Si **non** → on retourne **false**, car le chemin dans **Pieces** ne permet pas de battre le minotaure.

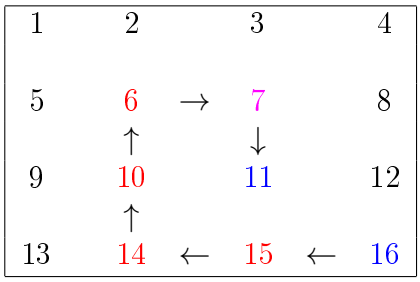
Oui, il est possible de vaincre le Minotaure, tweeter et sortir du labyrinthe avec une batterie de 15. (J'ai aussi pris en compte 5 de énergie pour la lumière).

Légende : salles visités | sortie/entree | Minotaure

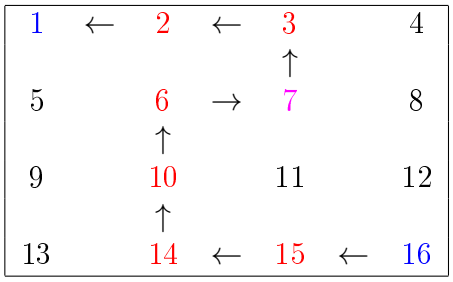
Pour une **Batterie = 14**, les chemins suivants sont possibles :

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | | 2 | | 3 | | 4 |
| | | | | | | |
| 5 | | 6 | → | 7 | | 8 |
| | | ↑ | | ↓ | | |
| 9 | → | 10 | | 11 | | 12 |
| ↑ | | | | | | |
| 13 | | 14 | | 15 | | 16 |

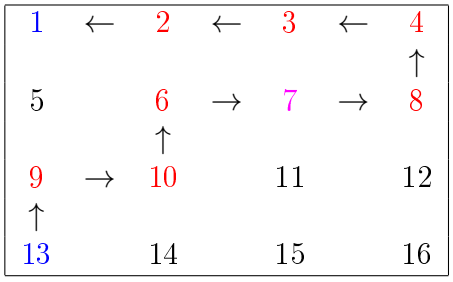



Pour une **Batterie = 15**, les chemins suivants sont possibles (les chemins **Batterie = 14** sont inclus) :




Pour une **Batterie = 16**, les chemins suivants sont possibles (les chemins **Batterie = 14 et 15** sont inclus) :



**.2)** Pour cette règle on fait exactement comme avant, pour trouver entree, sortie etc, la seule chose qui change, c'est qu'il faut modifier la règle **batterie(Pieces,Batterie,Reste)**, j'ai donc fait la règle **batterie_tweet(Pieces,Batterie,Reste)**, qui plutôt que les chemins qui ont un reste de énergie positive, elle accepte que les chemins qui donnent de l'énergie positive.

L'appel **reussite_tweet(15,Pieces)**, on a les résultats suivants :

(X -> symbolise le chemin no réalisable).

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | **X** | 2 | ← | 3 | ← | 4 |
| | | | | | | ↑ |
| 5 | | 6 | → | 7 | → | 8 |
| | | ↑ | | | | |
| 9 | → | 10 | | 11 | | 12 |
| ↑ | | | | | | |
| 13 | | 14 | | 15 | | 16 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | **X** | 2 | **X** | 3 | ← | 3 |
| | | | | | | ↑ |
| 5 | | 6 | → | 7 | → | 8 |
| | | ↑ | | | | |
| 9 | | 10 | | 11 | | 12 |
| | | ↑ | | | | |
| 13 | | 14 | ← | 15 | ← | 16 |