Rapport

Projet Prolog

Perret Quentin

Roche Eléa

[Dépot GitHub]

I. Choix du sujet

Notre sujet a pour objectif de répartir une population de pingouins dans le but d'organiser un tournoi de glissade sur la banquise du Pôle Nord.

Nous devons donc répartir nos pingouins en plusieurs groupes (les groupes allant de 1 à 6 pingouins). La répartition des pingouins dans les groupes n'est pas faite de manière aléatoire, en effet, il faut prendre en compte leur sexe, leur poids et taille ainsi que leur niveau de glissade pour que le concours soit fait de manière équitable.

De plus, il y a 3 pistes de glissade, qui ont des difficultés différentes. Les groupes de pingouins devront donc être répartis sur des pistes correspondant à leur niveau, afin de ne pas se mettre en danger, ou encore de ne pas s'ennuyer.

L'objectif de ce projet est donc de réaliser cette répartition en Prolog.

II. Mise en place du projet

Afin de réaliser ce projet, nous avons créé un dépôt Git afin de pouvoir se partager le code facilement, et de travailler simultanément sur deux ordinateurs différents.
Le code est disponible au lien suivant : https://github.com/QuentinPerret/Pingoui

De plus, avant de commencer la répartition en Prolog, nous avons choisi de réaliser un code Python sur un notebook afin de pouvoir générer automatiquement des pingouins et des pistes.
Le notebook penguin.ipynb contient donc :

- La génération de pingouins avec ce code :

```python
import itertools
import names
import numpy as np
import numpy.random as rd

class Pinguin:

    id_iter = itertools.count()

    def __init__(self) -> None:

        self.id = next(self.id_iter)

        self.sex = "female" if rd.randint(0,2) == 1 else "male"
        self.name = names.get_first_name(gender=self.sex)

        self.height = rd.randint(20,71)
        self.weigh = rd.randint(300,1000)
```

```
        self.level = rd.randint(1,4)
        self.last_position = rd.randint(1,7)

    def writePinguin(self):
        return
f"penguin({self.id},'{self.name}',{self.height},{self.weigh},{self.sex},{self.le
vel},{self.last_position}).\n"

N = 100

with open("penguin.pl","w+") as f:
    for i in range(N):
        P = Pinguin()
        f.write(P.writePinguin())
```

Si vous voulez changer le nombre de pingouins dans la population, il suffit de modifier la valeur de N.

- La génération de pistes :

```
import random
track_names = [
        "Mario Circuit", "Rainbow Road", "Bowsers Castle",
        "Yoshi Valley", "Wario Stadium", "Donut Plains",
        "Koopa Troopa Beach", "Luigi Raceway", "Toads Turnpike",
        "Mushroom Bridge", "Sherbet Land", "Delfino Pier",
        "Peach Beach", "Waluigi Stadium", "Shy Guy Beach",
        "DKs Jungle Parkway", "Baby Park", "Dry Dry Desert",
        "Tick-Tock Clock", "Airship Fortress", "Coconut Mall",
        "Maple Treeway", "Grumble Volcano", "Moonview Highway",
        "Mushroom Gorge"
    ]

class Slope:

    id_iter = itertools.count()
    unused_track_names = track_names.copy()

    def __init__(self, difficulty) -> None:

        if self.unused_track_names == []:
            Slope.unused_track_names = track_names.copy()

        self.id = next(self.id_iter)
        self.name = random.choice(self.unused_track_names)
        self.unused_track_names.remove(self.name)

        self.difficulty = difficulty

    def writeSlope(self):
        return f"slope({self.id},'{self.name}',{self.difficulty}).\n"
```

```python
def create_slope(num_pistes):
    with open("slope.pl", "w+") as f:
        for i in range(1,4):
            piste = Slope(i)
            f.write(piste.writeSlope())
        for i in range(num_pistes-3):
            difficulty = rd.randint(1, 4)
            piste = Slope(difficulty)
            f.write(piste.writeSlope())

create_slope(3)
```

Pour le bon fonctionnement de la répartition, il est important de garder que le nombre de piste créé est 3 (pour en avoir une de chaque difficulté)

Avant de lancer le code prologue, il faut donc lancer le code python se trouvant dans le fichier penguin.ipynb pour générer les fichiers penguin.pl et slope.pl contenant les éléments necessaires au bon fonctionnement des programmes distribution.pl et slope_distribution.pl .

III. Explication de la répartition

A. Les pingouins

1. Description de distribution.pl

Les pingouins sont représentés comme cela :
pinguin(id,'Nom',taille,poids,sexe,niveau,dernière position).
Le niveau d'un pingouin allant de 1 à 3 (1 étant expert, 2 intermédiaire, et 3 débutant), et sa dernière position de 1 à 6.

Objectif global: Nous avons donc choisi de créer des groupes de pingouins (allant de 1 à 6 pingouins) selon les critères suivants :

- **Regroupement par sexe**
  Les femmes étant ensemble, et les mâles ensembles.

- **Regroupement par IMC (Indice de Masse Corporelle)**
  Nous avons choisi de créer des groupes ayant des IMC inférieures à 1, des groupes ayant des IMC comprises entre 1 et 1,5, et des groupes ayant des IMC supérieurs à 1,5.
  Pour avoir l'IMC des pingouins, nous nous sommes servi de leur taille et de leur poids, grâce à la formule suivante :
  $IMC = \frac{Poids}{Taille^2}$
- **Regroupement par Niveau**
  Nous avons choisi que les groupes seraient composés de pingouins de même niveau (les niveaux 1 ensemble, les niveaux 2 ensemble et les niveaux 3 ensemble).
  Cependant, il y a une petite subtilité. En effet, nous nous sommes dit que les pingouins d'un

certain niveau, ayant pour dernière position la première place, devaient surement s'ennuyer. Nous avons donc décidé de les mettre avec les pingouins d'un niveau supérieur. Les pingouins de niveau 3 et de dernière position 1 sont donc dans des groupes de pingouins de niveau 2. De même, les pingouins de niveau 2 et de dernière position 1 sont donc dans des groupes de pingouins de niveau 1.

2. Détail de la stratégie

Chargement des données: Le code commence par charger les données des pingouins depuis un fichier `penguin.pl`. Ces données incluent des informations sur chaque pingouin, telles que son nom, sa taille, son poids, son genre, son niveau, etc.

```
% Retrieve penguins from penguin.pl file
?- write('\nLoading penguin.pl file ...\n'),
  consult('penguin.pl'),
  write('Loading Done !\n').
```

Génération de groupes par niveau:

- La règle `group_penguin_level/5` crée des groupes de pingouins en fonction de leur niveau.
- Elle récupère les pingouins qui n'ont pas remporté la dernière course à ce niveau, ainsi que ceux qui ont gagné la dernière course au niveau suivant (`has_won_last_race(P, LastPosition), NextLevel is Level + 1`).
- Les deux listes sont combinées, mélangées aléatoirement, puis divisées en groupes de taille spécifiée.

```
% Group by Level
group_penguin_level(Penguins, N, Gender, Level, Groups) :-
      % Get penguins at the specified level
      findall(P, (
      penguin(P, _, _, _, Gender, Level, LastPosition),
      \+ has_won_last_race(P, LastPosition)
      ), ListPenguins),
      % Get penguins who won the last race at the next level
      findall(P, (
      penguin(P, _, _, _, Gender, NextLevel, LastPosition),
      has_won_last_race(P, LastPosition),
      NextLevel is Level + 1
      ), WinnerPenguins),
      % Combine the two lists, randomize the order, and create groups
      append(ListPenguins, WinnerPenguins, AllPenguins),
      random_permutation(AllPenguins, RandomList),
      group(RandomList, N, Groups, Gender).
...
```

Génération de groupes par IMC et niveau:

- La règle `group_penguin_bmi_level/6` crée des groupes en fonction du niveau et de l'IMC.
- Elle utilise la règle `group_penguin_level/5` pour obtenir des groupes de pingouins au niveau spécifié.
- Ensuite, elle filtre ces pingouins en fonction de leur IMC dans une plage spécifiée.
- Les pingouins restants sont mélangés aléatoirement et divisés en groupes de taille spécifiée.

```
% Group by BMI and Level
group_penguin_bmi_level(Penguins, N, Gender, BMI1, BMI2, Level) :-
    % Group penguins by level
    group_penguin_level(Penguins, N, Gender, Level, LevelGroups),
    % Get penguins within the specified BMI range
    findall(P, (
        member(Group, LevelGroups),
        member(P, Group),
        penguin(P, _, Height, Weight, _, _, _),
        calculate_bmi(Weight, Height, BMI),
        BMI >= BMI1, BMI =< BMI2
    ), ListPenguins),
    % Randomize the order, and create groups
    random_permutation(ListPenguins, RandomList),
    group(RandomList, N, Penguins, Gender).
```

Création de tous les groupes:

- La règle `create_all_groups_bmi_level/2` créée tous les groupes en combinant les groupes pour différentes plages d'IMC et niveaux.
- Elle crée des groupes séparés pour les femelles et les mâles, puis les combine.

```
% Create groups by BMI and Level
create_all_groups_bmi_level(AllGroups, N) :-
    % Create groups for females
    create_groups_by_bmi_level(N, female, AllFemaleGroups),
    % Create groups for males
    create_groups_by_bmi_level(N, male, AllMaleGroups),
    % Combine female and male groups
    append(AllFemaleGroups, AllMaleGroups, ComplexAllGroups),
    % Combine groups for different levels
    append(ComplexAllGroups, AllGroups).
...

    % Combine groups for different BMI ranges and levels
    append([LowBmiGroups, MediumBmiGroups, HighBmiGroups, LowBmiGroups2,
MediumBmiGroups2, HighBmiGroups2, LowBmiGroups3, MediumBmiGroups3,
HighBmiGroups3], AllGroups).
```

Affichage des groupes:

- La règle `display_all_groups/1` est définie pour afficher toutes les informations sur les groupes générés.

- Elle affiche les détails de chaque pingouin dans chaque groupe, y compris leur nom, taille, poids, genre, niveau, position, et IMC.

Utilisation du code: L'utilisateur peut utiliser la règle `create_all_groups_bmi_level/2` avec un nombre spécifié de groupes (N) pour créer les groupes et les afficher.

```
% Code to use in a prolog terminal
?- create_all_groups_bmi_level(AllGroups, 5), display_all_groups(AllGroups).
```

B. Les pistes

1. Description de slope.pl

Et les pistes sont représentés comme cela :
slope(Id,'Nom',Difficulté).
La difficulté d'une piste allant de 1 à 3 (1 étant la plus dure et 3 la plus facile).

Objectif global: Ce code vise à distribuer des groupes de pingouins sur des pistes de ski en fonction du niveau du groupe. Les pistes ont des niveaux de difficulté correspondant aux niveaux des groupes.

2. Détail de la stratégie

Chargement des données: Le code commence par charger les données des pingouins depuis un fichier `distribution.pl` et les données des pistes depuis un fichier `slope.pl`.

```
% Retrieve penguins from distribution.pl file
?- write('\nLoading distribution.pl file ...\n'),
   consult('distribution.pl'),
   write('distribution.pl loading Done !\n').

% Load slopes from slope.pl file
?- write('\nLoading slope.pl file ...\n'),
  consult('slope.pl'),
  write('Loading slope.pl done !\n').
```

Affichage des pistes disponibles: Une règle (`display_slopes/0`) est définie pour afficher les pistes disponibles. Elle récupère la liste des pistes et affiche les détails de chaque piste, y compris son nom et sa difficulté.

Distribution des groupes sur les pistes:

- La règle `distribute_groups_on_slopes/1` prend une liste de groupes de pingouins en entrée.
- Pour chaque groupe, elle choisit un pingouin pour déterminer le niveau du groupe et calcule ce niveau en fonction du pingouin choisi.
- Ensuite, elle trouve une piste avec la difficulté correspondante au niveau du groupe.
- Elle affiche la distribution du groupe sur la piste et passe au groupe suivant.

```
% Rule to distribute groups on slopes
distribute_groups_on_slopes([]).
distribute_groups_on_slopes([Group | RestGroups]) :-
   % Choose a penguin from the group to determine the group's level
   member(P, Group),
   penguin(P, _, _, _, _, Level, LastPosition),
   % Calculate the group's level based on the chosen penguin
   (LastPosition \= 1 -> GroupLevel is Level; GroupLevel is Level - 1),
   % Find a slope with the corresponding difficulty
   slope(_, SlopeName, GroupLevel),
   % Display the distribution
   write('Group '), write(Group), write(' distributed on '), write(SlopeName)
,nl,
   % Distribute the rest of the groups
   distribute_groups_on_slopes(RestGroups).
```

Utilisation du code:

- L'utilisateur peut utiliser la règle `display_slopes/0` pour afficher les pistes disponibles.
- Ensuite, l'utilisateur peut créer des groupes de pingouins en utilisant la règle `create_all_groups_bmi_level/2`, et distribuer ces groupes sur les pistes en utilisant la règle `distribute_groups_on_slopes/1`.

```
% Code to use in a prolog terminal
?- display_slopes(),nl.
?- create_all_groups_bmi_level(AllGroups,6),
    distribute_groups_on_slopes(AllGroups).
```

IV. Prise de recul

Notre projet nous a permis de comprendre le fonctionnement et la manipulation de l'IA symbolique au travers du langage de programmation Prolog. Au travers de notre travail nous avons donc appris à définir ces règles et contraintes afin de réaliser une répartition logique de nos pingouins.
Ce langage a donc pour avantage de permettre la modélisation des problèmes complexes, en fournissant une représentation explicite des règles, des relations et des contraintes, et donc en simplifiant le processus de répartition.
De plus, nous avons constaté qu'il était facile de poursuivre le projet en permettant d'ajouter progressivement de nouvelles contraintes.

Cependant, l'utilisation de l'IA symbolique à des limites car elle implique d'avoir des données claires, précises et complètes.
De plus, nous nous sommes rendu compte tardivement que nous souhaitions ajouter une donnée dans les pingouins (la dernière position). Cela n'a pas été compliqué à rectifier car nos données étaient générées avec un code python, mais si cela n'avait pas été le cas, il aurait fallu les modifier manuellement ce qui est une grande perte de temps.
Enfin, si nous souhaitions agrandir le problème à très grande échelle, ou réaliser un traitement en temps réel, nous ne sommes pas certains que la solution trouvée soit satisfaisante, efficace et

rapide, et que les performances de notre programme soient optimales. Prolog n'étant peut être pas le langage à choisir pour de telles applications.

L'utilisation de l'IA symbolique et du langage de programmation Prolog nous a donc permis de réaliser un projet en facilitant le travail de représentation des règles et des contraintes, et donc de mener à bien ce projet à petite échelle, mais ces méthodes pourraient ne pas être efficaces à grande échelle et dans un temps raisonnable.