

Projet

Contexte

La société FunProg a décidé de développer une tondeuse à gazon automatique, destinée aux surfaces rectangulaires.

La tondeuse peut être programmée pour parcourir l'intégralité de la surface. La position de la tondeuse est représentée par une combinaison de coordonnées (x,y) et d'une lettre indiquant l'orientation selon la notation cardinale anglaise (N,E,W,S). La pelouse est divisée en grille pour simplifier la navigation. Par exemple, la position de la tondeuse peut être « 0, 0, N », ce qui signifie qu'elle se situe dans le coin inférieur gauche de la pelouse, et orientée vers le Nord.

Pour contrôler la tondeuse, on lui envoie une séquence simple de lettres. Les lettres possibles sont « D », « G » et « A ». « D » et « G » font pivoter la tondeuse de 90° à droite ou à gauche respectivement, sans la déplacer. « A » signifie que l'on avance la tondeuse d'une case dans la direction à laquelle elle fait face, et sans modifier son orientation. Si la position après mouvement est en dehors de la pelouse, la tondeuse ne bouge pas, conserve son orientation et traite la commande suivante. On assume que les cases directement au Nord et au Sud de la position (x, y) ont pour coordonnées respectivement (x, y+1) et (x, y-1).

Pour programmer la tondeuse, on lui fournit un fichier d'entrée construit comme suit : - La première ligne correspond aux coordonnées du coin supérieur droit de la pelouse, celles du coin inférieur gauche sont supposées être (0,0) - La suite du fichier permet de piloter toutes les tondeuses qui ont été déployées. Chaque tondeuse a deux lignes la concernant : - la première ligne donne la position initiale de la tondeuse, ainsi que son orientation. La position et l'orientation sont fournies sous la forme de 2 chiffres et une lettre, séparés par un espace - la seconde ligne est une série d'instructions ordonnant à la tondeuse d'explorer la pelouse. Les instructions sont une suite de caractères sans espaces.

Chaque tondeuse se déplace de façon séquentielle, ce qui signifie que la seconde tondeuse ne bouge que lorsque la première a exécuté intégralement sa série d'instructions. Lorsqu'une tondeuse achève une série d'instruction, elle communique sa position et son orientation. Le programme devra être en mesure de marcher avec un nombre non fixe de tondeuses. A la fin de l'exécution du programme, le résultat de l'exécution de toutes les tondeuses sera exportée en format json.

Objectifs et consignes

L'objectif de ce projet sera de concevoir et écrire un programme **Scala** implémentant la spécification ci-dessus.

De plus le dit porgramme devra respecter les consignes suivantes (**Zéro** sera donné en cas de non-respect de ses consignes):

- il est **INTERDIT** d'utiliser le mot clé `return`
- il est **INTERDIT** d'utiliser le mot-clé `while`
- il est **INTERDIT** d'utiliser le mot-clé `null`
- il est **INTERDIT** d'utiliser les expressions régulières
- tout `if` devra être **exhaustif** (c-à-d avec un `else`)
- tout **pattern-matching** devra être **exhaustif** (avec un cas par défaut sinon, ou sans warning si pas besoin de cas par défaut - avec `collect` par exemple -)
- la mutabilité explicite (`var`) ou implicite (structure de données mutables du package `scala.collection.mutable` comme `ArrayBuffer`) est **INTERDITE**
- seules les implémentations valides (qui compilent) seront prises en compte et seules celles renvoyant un résultat correct donneront lieu à la totalité des points

Un template de projet `sbt` sera fourni. Celui-ci servira de base pour les développements. Il n'est pas permis de changer les options de compilateurs définis pour ce projet. Lors du parsing des données d'entrées, si les données attendues ne sont pas fournies ou ne sont pas au format attendu, une exception de type `DonneesIncorectesException` (à créer) sera lancée avec un message d'erreur explicite. Il s'agit du seul type d'exception que le programme devra retourner. Veuillez bien à bien limiter les effets de bord aux frontières de votre programme (entrées/sorties) et non au coeur de celui-ci.

Entrées/sorties et Tests

Pour un exemple où le fichier suivant est fourni en entrée :

```
5 5
1 2 N
GAGAGAGAA
3 3 E
AADAADADDA
```

Les 2 tondeuses devront respectivement se terminer avec les positions suivantes:

```
1 3 N
5 1 E
```

On attend le résultat suivant (position finale des tondeuses) à la fin de l'exécution du programme:

```
{
  "limite": {
    "x": 5,
    "y": 5
  },
  "tondeuses": [
    {
```

```

    "debut": {
      "point": {
        "x": 1,
        "y": 2
      },
      "direction": "N"
    },
    "instructions": ["G","A","G","A","G","A","G","A","A"],
    "fin": {
      "point": {
        "x": 1,
        "y": 3
      },
      "direction": "N"
    }
  },
  {
    "debut": {
      "point": {
        "x": 3,
        "y": 3
      },
      "direction": "E"
    },
    "instructions": ["A","A","D","A","A","D","A","D","D","A"],
    "fin": {
      "point": {
        "x": 5,
        "y": 1
      },
      "direction": "E"
    }
  }
]
}

```

Version simplifiée

Cette section *simplifiée* est pour les groupes de 2 personnes.

Pour la version simplifiée du programme (voir la section Bonus), les données d'entrées pourront être fournies à votre guise: console (chaîne de caractères) ou fichier. De même le résultat de l'exécution de votre programme pourra être fourni selon votre préférence: console, fichier de sortie ... Si les entrées ou sorties sont fournies par fichiers, alors veuillez bien à ce que cela puisse être configurable. Pour cela un fichier de configuration sera fourni et permettra de

paramétrer le programme. Un template de fichier de configuration a été fourni et vous pourrez vous y référer (voir la section **Lecture des fichiers de conf** dans le **README** du projet).

Vous aurez dans le cadre de ce projet à définir des cas et données de tests qui permettront d'évaluer la correctitude de votre programme. Un guide pour écrire des tests automatisés (tests unitaires) sera fourni avec le projet.

Pour la sortie, en plus **json**, le programme devra en plus exporter sa sortie en **csv** et le chemin de la sortie devra être configurable. Vous aurez à déterminer la hiérarchie de **typeclasses** adéquates pour produire la sortie **csv**. L'utilisation de **typeclasses** est **OBLIGATOIRE** et la modélisation sera retenue dans la notation. Voici un exemple de la sortie pour l'exemple (de la section Entrées/sorties et Tests):

```
numéro;début_x;début_y;début_direction;fin_x;fin_y;fin_direction;instructions
1;1;2;N;1;3;N;GAGAGAGAA
2;3;3;E;5;1;E;AADAADADDA
```

Bonus

Cette section *bonus* est **OBLIGATOIRE** pour les groupes de 3 personnes. Ceci est un complément à la version simple.

Pour la sortie, en plus **json** et du **csv**, le programme devra en plus exporter sa sortie en **yaml** et le chemin de la sortie devra être configurable. Vous aurez aussi à déterminer la hiérarchie de **typeclasses** adéquates. L'utilisation de **typeclasses** est aussi **OBLIGATOIRE**. Voici un exemple de la sortie pour l'exemple (de la section Entrées/sorties et Tests):

```
limite:
  x: 5
  y: 5
tondeuses:
- debut:
  point:
    x: 1
    y: 2
  direction: N
  instructions:
    - G
    - A
    - G
    - A
    - G
    - A
    - G
    - A
```

```

- A
fin:
  point:
    x: 1
    y: 3
  direction: N
- debut:
  point:
    x: 3
    y: 3
  direction: E
instructions:
- A
- A
- D
- A
- A
- D
- A
- D
- D
- A
fin:
  point:
    x: 5
    y: 1
  direction: E

```

Evaluation

Pour les groupes de 2 et de 3, un barème commun sera utilisé et celui-ci prendra en compte les différences entre les sujets. Le barème d'évaluation sera le suivant:

- parseur pour données d'entrée / 4pts
- modélisation du moteur de gestion de la grille et des tondeuses / 6pts
- récapitulatif final (position des tondeuses) en `json`, `csv` et `yaml` (pour le bonus) / 8pts
 - pour les groupes de 2: `json`, `csv` (3pts et 5pts respectivement)
 - pour les groupes de 3: `json`, `csv` et `yaml` (2pts, 2pts et 4pts respectivement)
- scénarios de test / 2pts