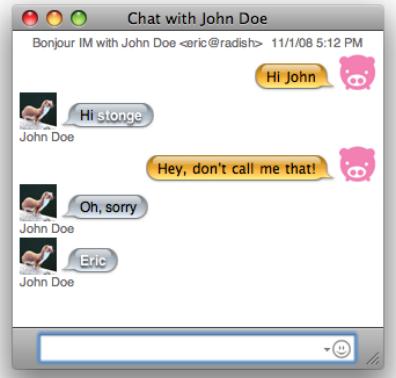
TP Avatar : premières fonctionnalités

1 Introduction

Au cours de ce TP vous allez développer une application répondant à des requêtes simples d'utilisateurs. Les requêtes, formulées en langage naturel, consistent en des recherches d'adresses sur Rennes. Cette application se présentera sous la forme d'un outil de messagerie instantanée à travers lequel l'utilisateur aura l'illusion d'interagir avec une personne réelle répondant à ses requêtes. Dans la suite, cette application sera nommée **avatar**. L'interface de l'avatar sera calquée sur l'interface d'un outil de messagerie instantanée, avec une zone de saisie de texte pour l'utilisateur et une zone de présentation graphique des échanges ayant déjà eu lieu.
Pour l'instant, votre client ne vous demande qu'une version rudimentaire de l'application, ne proposant que l'interface graphique et 3 fonctionnalités. Dans un prochain sujet, il vous demandera des extensions.



Préambule Le projet `Avatar` initial est disponible dans le Git : File>Import>Git

2 Contenu du projet `Avatar`

Le projet comporte plusieurs répertoires :

- Dans le répertoire `src`, vous trouverez un package :
  - ⋆ `machine` contient le trait `MachineDialogue` ainsi qu'une implémentation à compléter `MachineImpl` ;
  - ⋆ `client` contient une application de test automatique : l'objet `Client` testera votre implantation `MachineImpl` et vous dira si elle correspond aux attentes du client.
- Dans le répertoire `tests`, vous trouverez un package `machine` qui contient une première classe de test JUnit nommée `TestMachine` dans laquelle vous pourrez placer vos tests sur `MachineImpl`.
- Dans le répertoire `doc`, `DonneesInitiales.txt` définit la **syntaxe attendue** pour les noms de lieu et les adresses et dans `diag.uml` vous pourrez définir votre diagramme de classes.

3 Objectif du TP

L'objectif du TP est de développer une première version de l'avatar répondant à des requêtes simples à travers une interface graphique. Les fonctionalités F1, F2, et F3 attendues par le client sont les suivantes :

F1 L'avatar répondra à des requêtes concernant 4 lieux dans Rennes : le TNB, la mairie, la gare, et le théâtre la Paillette. Les noms et adresses devant être traités par l'avatar (et attendus par le client) sont donnés dans le fichier `donneesInitiales.txt` dans le répertoire `doc` du projet. Si la requête est incompréhensible par l'avatar, celui-ci le signale. Voici un exemple de dialogue attendu. Dans les réponses produites par l'avatar, les espaces sont représentés par un caractère '␣'. Pour que le client accepte votre programme, les réponses de l'avatar devront respecter **exactement** la ponctuation et les espaces entre les mots telle qu'elle est reproduite ici.

Utilisateur : `Je cherche la Mairie`

Avatar : `L'adresse␣de␣Mairie␣de␣Rennes␣est␣:␣Place␣de␣la␣Mairie`

Utilisateur : `et la Gare?`

Avatar : `L'adresse␣de␣Gare␣SNCF␣est␣:␣19,␣Place␣de␣la␣Gare`

Utilisateur : `je cherche`

Avatar : `Je␣ne␣comprends␣pas␣votre␣demande`

L'avatar associera certaines recherches à la même adresse. C'est le cas pour les recherches `Hôtel de ville`, `Mairie` et `Mairie de Rennes`, `TNB`, `Théâtre de Bretagne` et `Théâtre National de Bretagne`, `Gare` et `Gare SNCF`, `Théâtre la Paillette`, et `la Paillette`.

F2 L'avatar répond aux mêmes requêtes mais en tolérant certaines erreurs.

- L'avatar tolère l'oubli d'accents, majuscules, ou de mots de liaison.
- L'avatar accepte *au plus une erreur de frappe* par mot clé (une lettre soit manquante, soit erronée). On suggère ici d'utiliser la distance de Hamming (Google est votre ami) pour détecter une lettre erronée dans des mots de même taille. Attention, la distance de Hamming n'est définie que pour des chaînes de même longueur ! Pour détecter une lettre manquante, on pourra également se rapporter à la distance de Hamming entre le vrai mot clé et le mot entré au clavier par l'utilisateur augmenté (n'importe où) d'une lettre bien choisie.

Voici un exemple de dialogue attendu :

Utilisateur : `l'hotel de ville`

Avatar : `L'adresse␣de␣Mairie␣de␣Rennes␣est␣:␣Place␣de␣la␣Mairie`

Utilisateur : `hotl de valle`

Avatar : `L'adresse␣de␣Mairie␣de␣Rennes␣est␣:␣Place␣de␣la␣Mairie`

Utilisateur : `l'hot de ville`

Avatar : `Je␣ne␣comprends␣pas␣votre␣demande`

F3 L'interaction avec l'avatar est humanisée. Si la requête de l'utilisateur contient les mots `bonjour`, `salut`, `bonsoir` (modulo quelques petites erreurs, voir F2), l'avatar commence sa réponse par `Bonjour` (dans une bulle différente). Voici un exemple de dialogue attendu :

Utilisateur : `Bonjour, ou se trouve la Mairie?`

Avatar : `Bonjour`
`L'adresse␣de␣Mairie␣de␣Rennes␣est␣:␣Place␣de␣la␣Mairie`

Utilisateur : `Bonjour`

Avatar : `Bonjour`

Utilisateur : `Bonjour, comment tu t'appelles?`

Avatar : `Bonjour`
`Je␣ne␣comprends␣pas␣votre␣demande`

4 Méthodologie

Equipes suggérées : Base de données, Interface graphique, Analyse de la phrase (recherche de mots clés), Tolérance aux fautes, Tests, Avatar (application)

Test : On vous conseille d'utiliser une approche dirigée par les tests (TDD) pour le développement de ce projet.

Intéraction avec le client : Pour pouvoir interagir avec le client, votre application doit implémenter les opérations `reinit:Unit` et `test(l:List[String]):List[String]` du trait `MachineDialogue`. Pour tester votre avatar, le client le reinitialisera en utilisant `reinit` et vérifiera les interactions avec `test`. La liste en entrée de `test` donne la liste des questions posées par l'utilisateur. La liste en sortie donnera la liste de réponses produites par l'avatar. Par exemple, le résultat attendu pour :

`test(List("Bonjour", "ou se trouve la mairie?", "scsdgar??"))` est

`List("Bonjour","L'adresse␣de␣Mairie␣de␣Rennes␣est␣:␣Place␣de␣la␣Mairie","Je␣ne␣comprends␣pas␣votre␣demande")`