Université “Pierre et Marie Curie”

PC2RON

BUNDO Eni BELKAID Mehdi

December 19, 2011

Abstract

Dans le cadre du projet PC2R, notre travail consiste à réaliser le jeu PC2RON en utilisant une architecture Client/Serveur programmée en deux langages différents et respectant un certain protocole au niveau des échanges entre des Clients dont la tâche principale consiste à gérer les actions des joueurs, et un serveur qui joue le rôle de l'arbitre du jeu.

Contents

1 Serveur

1.1 Choix du langage

Dans cette partie, nous avons choisi le langage Python pour la réalisation du Serveur. Python est un langage de programmation facile à utiliser et dispose d'un grand nombre de bibliothèque, notamment "twisted" nous permettant dans le cadre de notre projet un gain de temps considérable dans la programmation du serveur.

1.2 Libraries utilisées

1.2.1 Twisted

Cette librarie nous donne une structure prédefinie du serveur. Pour notre jeux nous avions besoin d'un protocol *stateful* c'est à dire un protocol à état (état est une paire *(fonction, numOctets)*. Lorsque *numOctets* octets de données arrivent à partir du réseau, la *fonction* est appelée. Il est prévu de retourner l'état suivant ou None (*null* en python) pour maintenir un même état. L'état initial est retourné par la fonction *getInitialState()*.).
Une instance de la classe MyProto est créée pour chaque connection au serveur, c'est à dire une instance de la classe MyProto pour chaque joueur. Par contre il y a qu'une seule Factory pendant l'execution du serveur, donc tous les éléments partagés (comme par exemple la matrice du jeux) doivent se trouver dans la Factory, pour que tous les joueurs puissent y accéder.

1.2.2 Struct

Cette librarie est utilisée pour effectuer des conversion entre des valeurs Python vers des structures C représentées comme des chaînes Python. Voici un exemple d'utilisation:

```
>>> struct.pack('!h', 42)
'\x00*'
>>> struct.unpack('!h', '\x00*')[0]
42
```

On a utilisé cette librarie pour la conversions des trames et des types de données dans les modules trames.py et aussi cons.py.

1.3 trames.py

Dans ce module les trames *"statiques"* sont definies. La structure choisie pour une trame est une liste ou le premier élément c'est le id de la trame et le deuxieme élément c'est une autre liste contenant des couples, ou chacun correspend à la représentation d'une donnée. Voici une ligne extraite du code du serveur qui montre cela:

```
sendFrame(self, [0x43, [("string", "NO"), ("string", "No more place!")]])
```

Ici on envoie la trame d'id 0x43 qui contient deux champs de données, le string "NO" et le string "No more place!".
La structure générique de la trame est de la forme suivante

```
[id, [(type_1, val_1),...,(type_n, val_n)]]
```

Ou *type_i* prend une des valeurs suivantes : {"int8", "uint8", "int16", "uint16", "int32", "uint32", "double", "string"} et *val_i* correspond à la valeur da le i-eme donnée.

1.4 cons.py

Dans ce module on s'occupe de la construction d'une trame (conversion d'une trame de notre structure en une chaîne python contenant tous les octets de la trame).

```
>>> import cons
>>> cons.constructFrame([0x42,
... [(‘‘uint16’’, 42),
...  (‘‘string’’, ‘‘foo’’),
...  (‘‘double’’, 1.337),
...  (‘‘string’’, ‘‘bar’’)]])
’\xffB\x04\x12\x00* \x00\x03foo0?\xf5dZ\x1c\xac\x081 \x00\x03bar’
>>>
```

Cette valeur est prete à etre envoyée sur le reseau aux clients.

1.5 gamemodule.py

Ce module contient la classe Game, l'implementation du jeux. Une instance de cette classe contient les references de tous les joueurs et aussi une matrice qui joue le role du terrain. La *"boucle principale"* de cette classe est la fonction gameLoop():

```
def gameLoop(self):
    if self.stillPlaying:
        self.instant()
        reactor.callLater(0.035, self.gameLoop)
    else:
        self.sendWinner()
```

Aussi les fonctions qui gérent les changements de direction, de position ainsi que les collisions, se trouvent dans ce module.

1.6 user.py

Ce module contient la classe User. Chaque instance de cette classe contient les champs néccessaires pour gérer les joueurs pendant le jeux. *Quatre* instances de cette classe se trouvent dans Game au debut du jeux, chaque une correspond a un joueur. Un champ appelé *protocol* se trouve dans la classe User, celui-ci est de type *MyProto* qui est defini dans la classe server_sync.py, et qui nous permet d'envoyer des trames aux joueurs.

2 Client

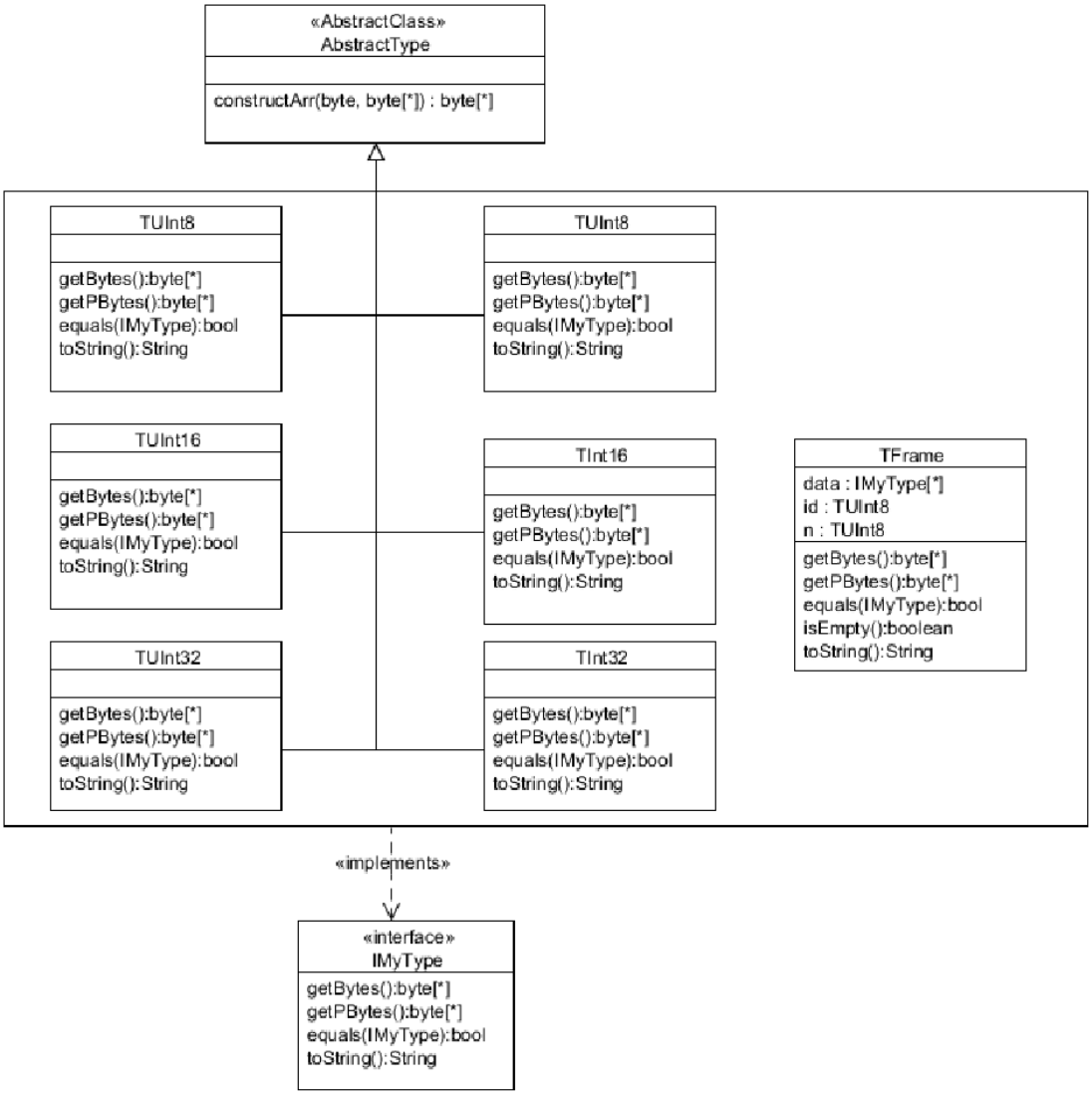
2.1 Choix du langage

Dans cette partie, nous avons choisi le langage JAVA pour la réalisation du Client. En effet, l'existence des classes prédéfinies Socket (pour l'utilisation d'un socket client en mode TCP) ainsi que les classes DataOutputStream et DataInputStream pour la lecture et l'écriture des flux d'entrée/sortie associés au socket Client et offrant la possibilité de lire directement des types primitifs (double, char, int, byte), nous facilite la gestion fastidieuse des trames échangées avec le serveur. D'autre part, étant donne que l'interface du joueur est écrite coté Client, JAVA offre une panoplie de classes prédéfinies pour faciliter sa programmation.

2.2 Package types

Le package types se constitue des classes qui suivent:

- *TInt8.java* - pour représenter les entiers signés tenant sur 8 bits
- *TUInt8.java* - pour représenter les entiers non signés tenants sur 8 bits
- *TInt16.java* - pour représenter les entiers signés tenant sur 16 bits
- *TUInt16.java* - pour représenter les entiers non signés tenants sur 16 bits
- *TInt32.java* - pour représenter les entiers signés tenant sur 32 bits
- *TUInt32.java* - pour représenter les entiers non signés tenants sur 32 bits
- *TDouble.java* - pour représenter les doubles
- *TString.java* - pour représenter les strings
- *TFrame.java* - pour représenter une trame entiere contenant plusieurs données



2.3 Package tools

Dans ce package on peut trouver les classes:

- ArrayTools.java - Pour concatener deux arrays de bytes, et pour comparer des arrays
- FrameAnalyser.java - Cette classe contient toutes les trames predefinies comme par example

```
{ 0x49, string ‘‘PC2RON?’’, string ‘‘PC2RON2011’’ }
```

```
{ 0x4F, string ‘‘left’’ }
```

- MyParser.java - Cette classe connait le flux d'input (DataInputStream). La methode $parseFrame()$ appelle $parseData()$ n fois ou n est le nombre de données (data dans le code source) qui se trouvent dans la trame, cette derniere va apres *dispatcher* les appels au bon endroit ($parseInt8$, $parseUInt8$, etc.) d'apres le type des données.

3 Tests et conclusion

Après plusieurs jeux de tests réalisés sur notre application, nous nous sommes assurés du bon fonctionnement de la partie pour laquelle nous avons accordé la majorité du temps de programmation, en l'occurrence la version avec état global réalisant tous les échanges nécessaires au bon déroulement d'une partie.

De plus, la mise au point d'une interface graphique nous permet de confimer le bon fonctionnement de l'application en multijoueurs.

Enfin ce projet nous a permis d'une part d'approfondir nos connaissances en langage de programmation grâce à la diversité des composants utilisés, et d'autre part une bonne maitrise de la programmation concurrente réactive et répartie.