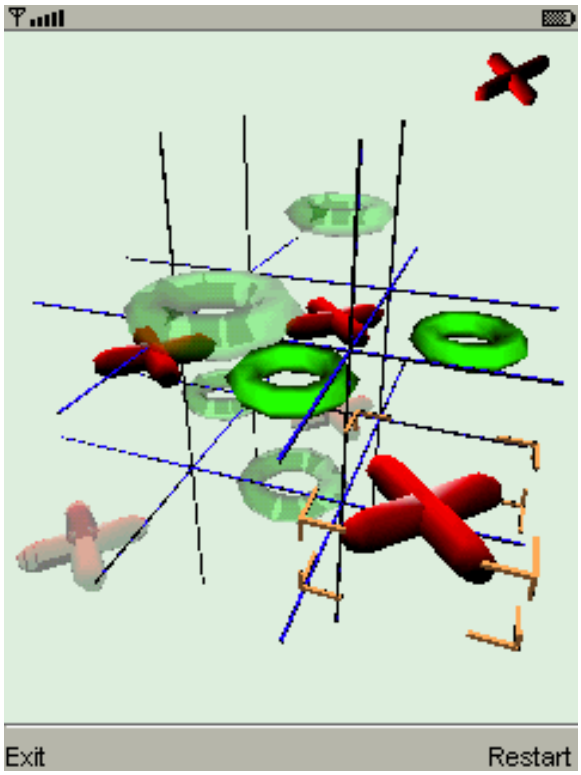
Béatrice Frey et Cyril Crassin **Printemps 2005**

Projet LO52

Tic Tac Toe

J2ME Bluetooth



Sommaire

Introduction

Le but de ce projet était de développer un jeu de Tic Tac Toe sur plateforme j2me capable de fonctionner entre deux téléphones mobiles, PDA, ou tout autre périphérique portable. Le jeu devait mettre en place une communication *Bluetooth* entre les périphériques afin de permettre aux joueurs de jouer l'un contre l'autre.

Le développement de l'interface graphique était libre, après un certain nombre de recherches pour savoir comment nous allions procéder, nous avons découvert l'API Mobile 3D pour j2me. Cette API graphique permet d'insérer de la 3D temps réel dans les applications j2me et est même accélérée matériellement sur un certain nombre de périphériques. Etant tout les deux bien intéressé par ce genre de possibilité, nous avons décidé de nous lancer dans le développement d'une interface graphique en 3D pour notre jeu.

1. La plateforme J2ME

1.1 Présentation

J2ME est la plate-forme Java pour le développement d'applications sur appareils mobiles tels que les PDA, téléphones portables, systèmes de navigations pour voiture, etc. C'est une sorte de retour aux sources puisque Java avait été initialement développé pour piloter des appareils électroniques.

Historiquement, J2ME n'est pas la 1ere plateforme proposée par SUN pour le développement d'applications sur des machines possédant des ressources réduites, typiquement celles ne pouvant exécuter une JVM répondant aux spécifications complètes de la plate-forme J2SE. Avant J2ME il y avait en effet :

- *JavaCard* : pour le développement sur des cartes à puces.
- *EmbeddedJava* : pour le développement embarqué.
- *PersonnalJava* : pour le développement sur des machines possédant au moins 2mo de mémoire.

En 1999, Sun décide de mieux structurer ces différentes plate-formes sous l'appellation J2ME (Java 2 Micro Edition). Seule le plate-forme JavaCard n'est pas incluse dans J2ME et reste à part. Par rapport à J2SE, le propre de J2ME est d'utiliser des machines virtuelles différentes et allégées, un grand nombre de classes de base étant communes aux deux API.

1.2 Les configurations et profils

L'ensemble des appareils sur lequel peut s'exécuter une application J2ME est tellement vaste que l'API doit proposer une architecture modulaire, composée de plusieurs parties : les *configurations* et les *profils* qui sont spécifiés par le JCP (*Java Community Process*). Chaque configuration peut être utilisée avec un ensemble de packages optionnels qui permet d'utiliser des technologies particulières telles que le Bluetooth ou la 3D qui nous intéressent mais également les services web, lecteur de codes barre, etc… Ces packages sont le plus souvent dépendant du matériel.

Toutes les mises à jour de J2ME, que ce soit des configurations ou des profils, sont définies dans des JSR (*Java Specification Request*) qui permettent d'étendre la spécification originale de l'API.

Les *configurations* définissent les caractéristiques de bases d'un environnement d'exécution pour un certain type de machine possédant un ensemble de caractéristiques et de ressources similaires. Elles se composent d'une machine virtuelle et d'un ensemble d'API de base. Deux configurations sont actuellement définies : CLDC *(Connected Limited Device Configuration)* et CDC *(Connected Device Configuration)*.

Les *profils* se composent d'un ensemble d'API particulières à un type de machines ou à une fonctionnalité spécifique. Ils permettent l'utilisation de fonctionnalités précises et doivent être associés à une configuration. Ils permettent donc d'assurer une certaine modularité à la plate-forme J2ME.

L'inconvénient de cette architecture est qu'elle permet le développement d'applications pour un profil particulier ce qui déroge au principe initial de Java qui était de permettre l'écriture d'applications fonctionnant de manière identique et sans adaptation sur toutes les plateformes supportées. Mais la diversité des architectures matérielles cibles de l'API oblige à ce type de fonctionnement qui permet tout de même d'accéder à des fonctionnalités propres au matériel à condition de fixer un profile minimum pour l'application.

1.3 MIDP

MIDP (Mobile Information Device Profile) est un profil pour les appareils mobiles utilisant la configuration CLDC, comme les téléphones mobiles. Il s'agit d'une API Java de haut niveau permettant la gestion de l'interface utilisateur (GUI), la gestion de l'interface réseau, le stockage local de données ainsi que la gestion d'une base de donnée sur le mobile et la supervision d'applications.

MIDP est le premier profil qui a été développé dont l'objectif principal est le développement d'applications sur des machines aux ressources et à l'interface limitées tel qu'un téléphone cellulaire. Ce profil peut aussi être utilisé pour développer des applications sur des PDA de type Palm.

C'est MIDP qui définit les applications de type *midlets* que nous avons utilisé pour ce projet.

2. L'API Bluetooth

2.1 Présentation

L'API *Bluetooth* pour J2ME est définie dans la JSR-82 et a été conçue pour fonctionner sur configuration CLDC en surcouche au profile MIDP dont elle utilise le *framework* générique de connexion. Elle définit deux packages: *javax.bluetooth* qui contient le corps de l'API Bluetooth de communication et *javax.obex* pour le protocole d'échange d'objets OBEX (*Object Exchange*).

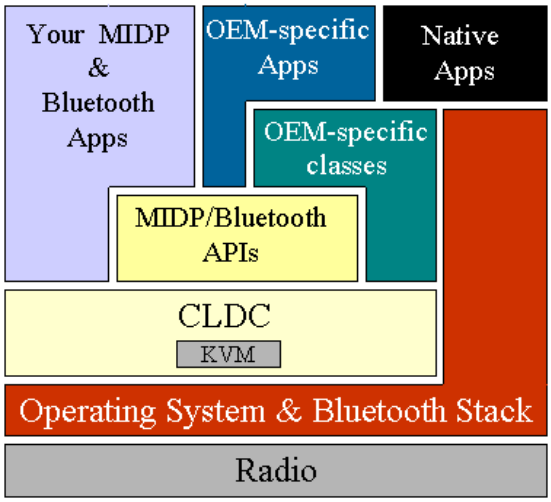


Fig. 1 : Localisation de l'API bluetooth dans l'architecture J2ME et interaction avec le système

2.2 Fonctionnement

L'API *Bluetooth* fonctionne sur le modèle client/serveur. Un serveur crée et déclare des services auxquels les clients peuvent venir se connecter afin d'engager une communication.

Une communication *Bluetooth* se décompose donc en 5 parties : l'initialisation de la pile *Bluetooth*, la gestion et la configuration des périphériques, la découverte des périphériques serveurs pour les clients, la découverte de services sur le périphérique serveur, et enfin la communication en elle-même.

3. L'API Mobile 3D

L'API Mobile 3D définie dans la JSR-184 apporte le support du graphisme 3D à J2ME. Il s'agit tout comme l'API *bluetooth* d'un package optionnel pour CLDC 1.1 qui se place en surcouche de MIDP. L'API définit des interfaces de programmation de bas et de haut niveau pour l'affichage graphique 3D sur des périphériques dotés de peu de mémoire et d'une faible puissance de calcul qui ne sont pas forcément équipés d'une accélération 3D matérielle. Mais elle est suffisamment flexible et extensible pour s'adapter également aux nouveaux périphériques dotés d'écrans couleur, d'accélération 3D ou du support des opérations en virgule flottante.

L'API propose deux modes de fonctionnement :

- Un *mode immédiat* de bas niveau (*immediate mode*) qui fonctionne sur le même principe qu'OpenGL ou que Direct3D et qui s'aligne sur le standard fixé par OpenGL ES (*OpenGL for Embedded Systems*). Ce mode permet le dessin direct d'objets et toute l'organisation de la scène doit être faite par l'utilisateur.
- Un *mode retenu* de plus haut niveau (*retained mode*) qui fonctionne un peu sur le même principe que Java 3D et qui permet de travailler sur un graphe de scène.

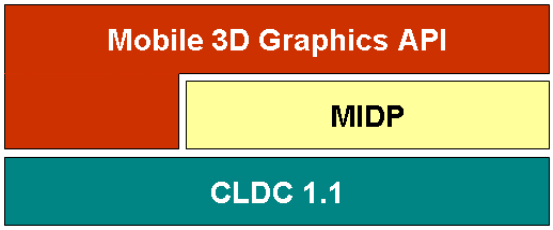


Fig.2 : Intégration de l'API Mobile 3D dans l'architecture J2ME

4. L'application Tic Tac Toe

Notre application se base sur le profile MIDP 2.0 qui permet d'utiliser les API *bluetooth* et *Mobile 3D* ainsi que sur la configuration CLDC 1.1 afin de pouvoir utiliser les nombres à virgule flottante qui ne sont pas supportés en CLDC 1.0.

L'application fonctionne sur le modèle client/serveur, il s'agit du modèle générique d'une application *bluetooth*. Au début du jeu, chaque joueur choisit son numéro (joueur 1 ou 2) et cela détermine son rôle dans le modèle de communication *bluetooth*. Le joueur 1 joue en effet le rôle de serveur qui crée le service de jeu et se met en attende d'une connexion d'un joueur 2.

4.1 Structure de l'application

Notre application se découpe en trois grandes parties : la visualisation et la gestion du jeu, la communication et la classe principale chargée d'instancier les objets et de lancer le jeu.
La communication a été abstraite dans une classe appelée *TicTacCom* qui offre une interface de communication au module de jeu. Cette classe gère toute la mise en place de la communication entre les deux joueurs et son interface permet l'envoi d'un coup et la réception du coup de l'adversaire.

La gestion du jeu et l'affichage se trouvent dans la classe TicTacCanvas qui se charge de rendre l'interface graphique en 3D, de récupérer les actions de l'utilisateur et fixe la logique du jeu. Le rafraîchissement de l'affichage est commandé via un timer appelé *MyTimerTask* qui appel la méthode de dessin à intervalles réguliers.

La classe principale de l'application est le *MIDlet* lui-même qui sert de support à l'application embarquée.

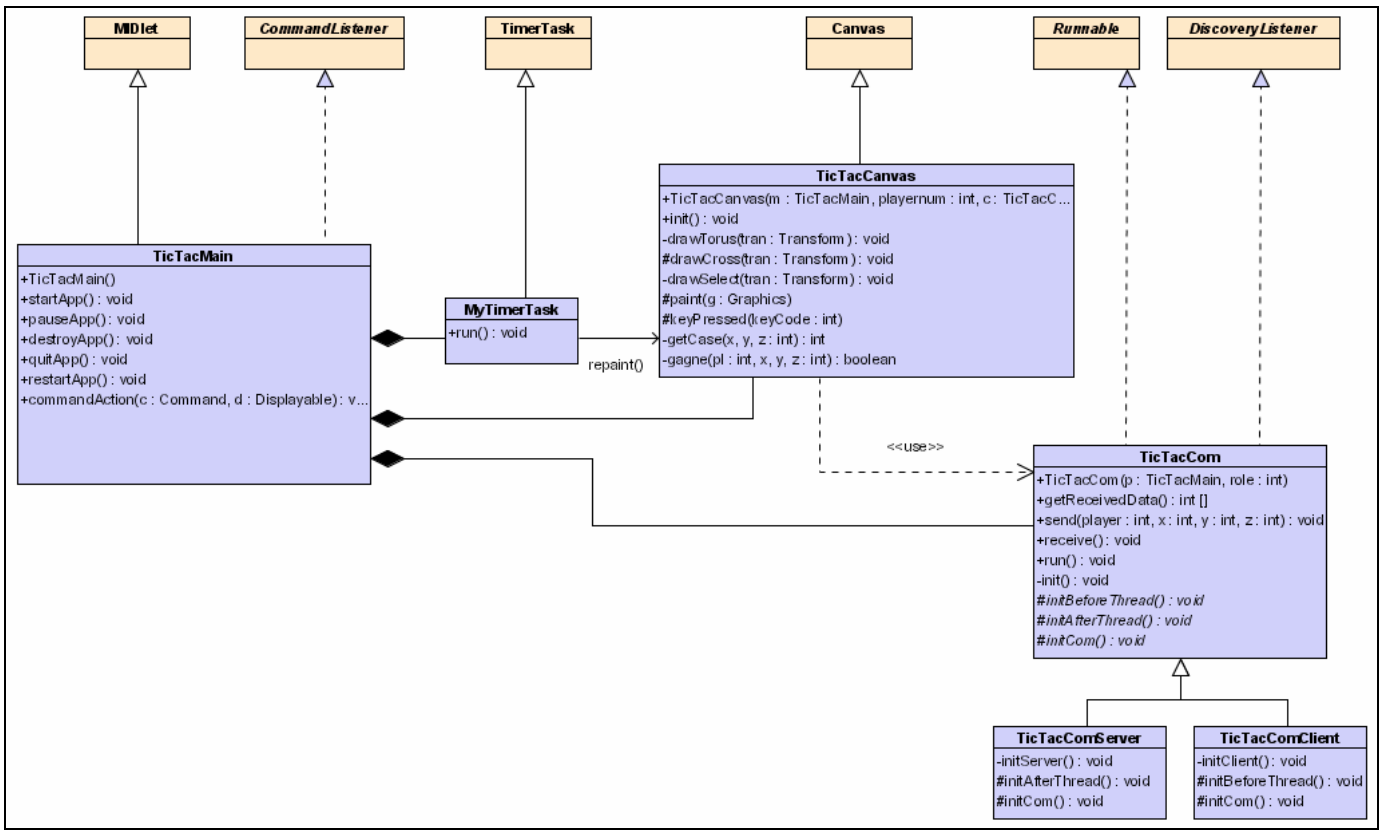


Fig.3 : Diagramme de classes général de notre application.

4.2Le module de communication

Le module de communication est le premier à être initialisé par la classe *TicTacMain*. Son comportement général, commun aux deux joueurs, se différencie en deux sous classes appelées *TicTacComServer* et *TicTacComClient* (cf. *fig. 3*) qui permettent d'initialiser la communication spécifiquement au rôle de communication du joueur. Le comportement du jeu étant totalement symétrique pour les deux joueurs, ce rôle a été fixé arbitrairement, le joueur 1 étant le serveur et le joueur 2 le client. Une fois la communication établie, le comportement de la classe est exactement le même pour les deux joueurs et toutes les méthodes de communication d'informations sont donc communes.

Cette distinction de l'initialisation de la communication est rendue nécessaire par le mode Client/Serveur adopté par la norme *bluetooth*. Le fonctionnement de ce module est totalement désynchronisé du jeu lui-même par *threading* ce qui permet d'effectuer des envois et d'attendre des messages de manière non bloquante en ne figeant pas l'interface graphique.

4.2.1 Initialisation du serveur

Le serveur *bluetooth* à pour rôle de fournir un « service » qui pourra être découvert et utilisé par un client. La première étape de cette initialisation consiste à récupérer le périphérique local (*device)* de communication *bluetooth*. Il faut ensuite rendre se *device* « découvrable » pour permettre aux clients de le trouver lors d'une recherche de périphériques *bluetooth* distants.

Un nouveau service est ensuite ajouté à ce périphérique via une adresse de connexion (URL) composée d'un identifiant (*UUID*) partagé par le serveur et le client et qui permet d'identifier le client. L'adresse contient également le profile communication utilisé, il s'agit ici de *SPP* (*Serial Port Profile*) qui utilise le protocole *RFCOMM* et qui offre un comportement semblable à celui d'un port série. Ce type de protocole permet une fois la communication établie, lors de l'acceptation d'une demande entrante d'un client, de communiquer simplement par flux dans les deux sens.

Toute la communication se fait ensuite de manière transparente par rapport au *bluetooth* via le *frameworks* de communication générique de CLDC.

4.2.2 Initialisation du client

L'initialisation de la communication coté client est légèrement plus complexe. Elle consiste tout d'abord à récupérer sur le périphérique local un agent de découverte (*discovery agent*) et à initier une recherche de terminaux mobiles disposant d'une interface *bluetooth* dans la zone de couverture. Parmi les terminaux trouvés à l'étape précédente, le client recherche alors ceux offrant le service nécessaire.

Une fois que le service qui nous intéresse est découvert, son URL est récupérée et la communication est établie. Des flux d'entrées et de sorties sont alors mis en place et les communications peuvent commencer.

4.2.3 Echange de données

Les données échangées entre les deux joueurs sont simplement des coups constitués des coordonnées X, Y et Z sur le plateau de jeu. Les communications étant non bloquantes, des méthodes de test de réception de message sont fournies. Une boucle de communication permet d'envoyer les nouveaux messages placés dans un *buffer* synchronisé par le *thread* de jeu local et de recevoir les messages venant de l'autre joueur. Une attention particulière a du être portée à la synchronisation des données entre les deux *threads* (jeu et communication). Toutes les communications sont font par lectures et écritures d'entiers dans des flux d'entrée et de sortie.

4.3Le module de jeu et l'affichage

Le module de jeu a été implémenté dans la classe *TicTacCanvas* (cf. *fig. 3*). Sa première fonction est d'initialiser l'interface graphique et notamment l'affichage 3D. Lors de cette étape, on commence par créer le menu qui permettra de quitter et de redémarrer le jeu, un objet Graphics3D est ensuite récupéré et servira de point d'entrée pour toutes les fonctions d'affichage 3D.

4.3.1 Affichage 3D

Afin de pouvoir optimiser au maximum cet affichage, le mode immédiat a été choisit, ce mode est en effet de très bas niveau et permet un contrôle total sur les opérations effectuées. Lors de l'étape d'initialisation les différents objets 3D (Tore, croix, curseur et plateau de jeu) sont chargés à partir d'un fichier m3g. Ce format de fichier 3D a été conçue spécialement pour Mobile3D et est extrêmement compact et adapté aux contraintes des périphériques mobiles. Nous avons réalisé la modélisation de ces objets via 3D studio MAX qui offre maintenant la possibilité d'exporter dans ce format. Le point de vue et la source de lumière sont ensuite définis et un maximum de transformations et de propriétés liés aux objets sont précalculés afin d'alléger au maximum la boucle de rendu.

A intervalle régulier, la méthode de dessin est appelée. C'est elle qui permet de redessiner l'interface. Les rotations de la scène en fonction de la position du curseur sont alors calculés et l'ensemble du plateau de jeu avec ses pions sont rendus. Une attention toute particulière a été portée à cette étape afin d'optimiser au maximum les opérations réalisées et rendre ainsi l'affichage exploitable sur un matériel aux ressources limitées.

4.3.2 Gestion du jeu

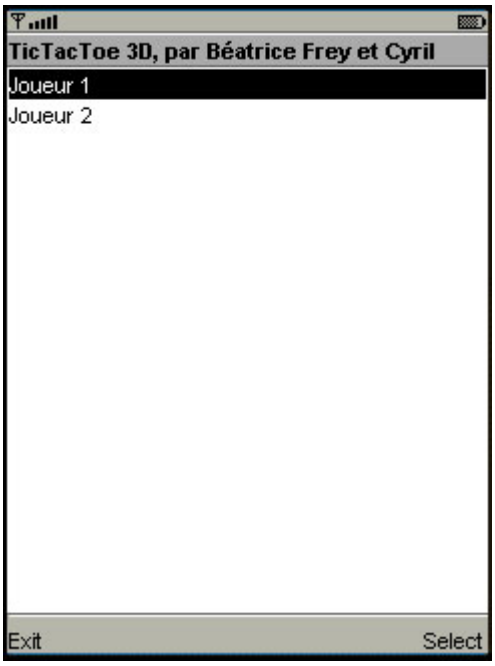
Le plateau de jeu est initialisé dans l'étape d'initialisation. Ensuite, le jeu peut se trouver dans 3 états : « tour du joueur local », « en attente du coup de l'autre joueur » et « partie terminée ». A chaque rafraîchissement de l'image, l'interface est dessinée en fonction de cet état. Si on est en attente du coup de l'autre joueur, on test si le coup nous est parvenu via le module *TicTacCom* et on adapte l'interface. Si c'est à nous de jouer, le curseur peut être déplacé en récupérant les évènements clavier via la méthode *keyPressed* surchargée du *Canvas*. Le code de la touche est alors traduit en action via la méthode *getGameAction* qui permet de rendre le code indépendant des interfaces d'entrée utilisées. Ces interfaces pouvant être différentes en fonction du type de matériel sur lequel l'application est exécutée. Lorsque le joueur place son pion, le coup est envoyé par le module *TicTacCom*. On test alors si le coup donne la victoire au joueur et si c'est le cas, on passe dans l'état « partie terminée » et un sous état « victoire » est activé. L'interface s'adapte alors en conséquence. Lorsque l'autre joueur reçoit le coup, il test lui aussi si le coup a donné la victoire à son adversaire et adapte ses états « partie terminée » et « défaite » en conséquence.

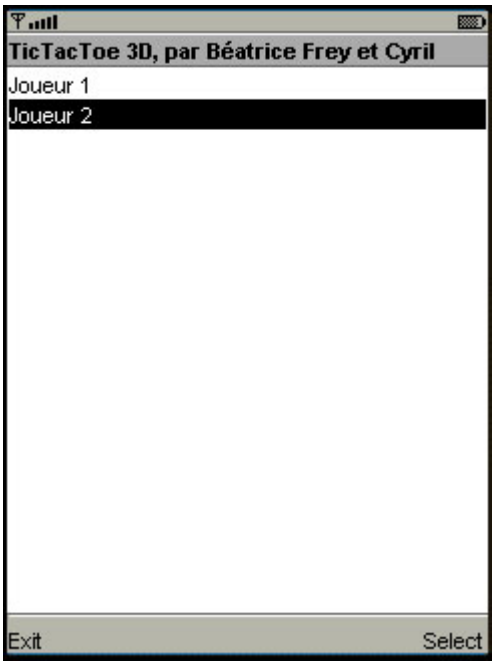
4.3.3 Gestion des commandes

Afin de garantir une bonne navigation de l'utilisateur dans les différentes interfaces proposées, nous avons mis en place différentes commandes. Des commandes simples pour naviguer dans le menu ont été réalisées avec l'appel à *commandAction* qui prend notamment en paramètre un objet de type *Command*. Ces commandes permettent à l'utilisateur de sélectionner son numéro de joueur (*Select*), de relancer une nouvelle partie (*Restart*) ou encore de quitter l'application (*Exit)*. De la même manière, des contrôles ont été créés pour permettre à l'utilisateur de positionner son pion. Ces commandes font appel à la fonction *keyPressed* qui teste la touche sur laquelle l'utilisateur à appuyer.

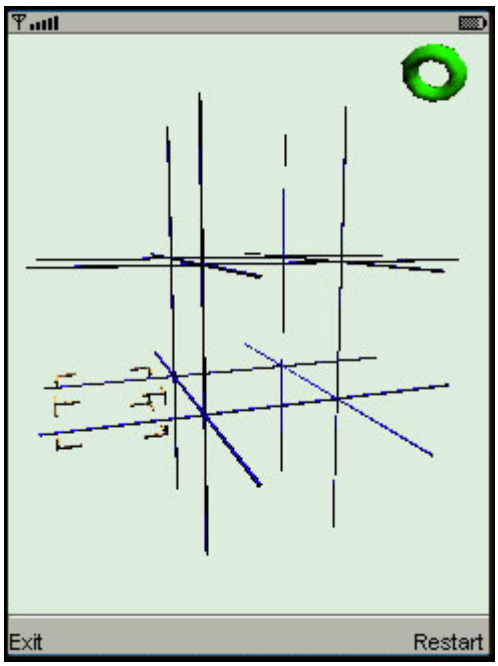
5. Déroulement du jeu

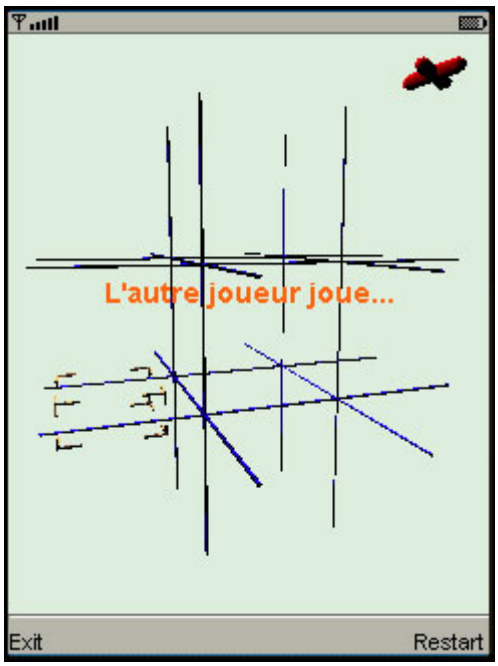
Au lancement de l'application, chaque utilisateur doit choisir un joueur. Le joueur 1 initialisant le serveur *bluetooth*, il doit impérativement être choisit en premier, c'est-à-dire avant que le joueur 2 n'ait validé son choix.



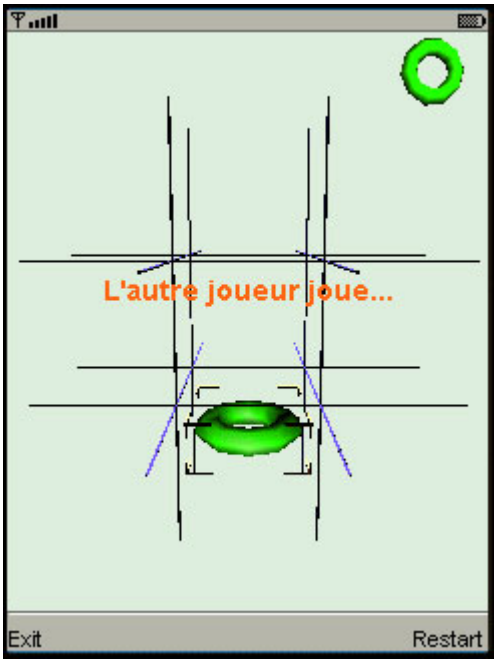


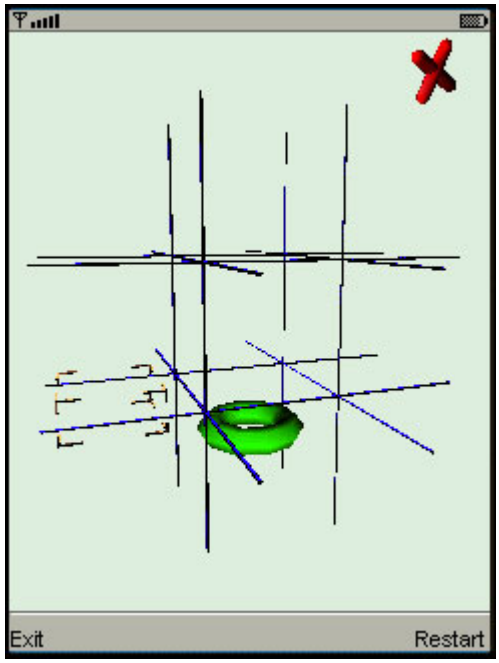
Une grille de jeu s'affiche alors aux utilisateurs. Ici, le joueur 1 joue avec les ronds, le joueur 2 avec les croix. Le joueur 1 joue en premier, il peut donc, en appuyant sur les touches de déplacement, placer son pion sur la grille. Le déplacement de case en case est visualisé par un curseur. Le joueur 2, lui, est en position d'attente. Jouant en second, il doit attendre que le joueur 1 ait placé son pion. Un message d'attente s'affiche et l'interface se fige pour que l'utilisateur n'essaie pas de jouer en même temps.





Lorsque le joueur 1 a placé son pion, ce dernier s'affiche sur son interface. L'information est également envoyée au joueur 2 pour que celui-ci visualise également le pion placé. Les rôles sont alors inversés, le joueur 1 passe en attente et le joueur 2 peut à son tour placer un pion sur la grille.





La partie continue ainsi plusieurs tours. Deux cas se présentent alors, soit il y a ex æquo et l'application l'indique par un message, les joueurs peuvent alors recommencer une partie en appuyant sur *Restart*, soit un des joueurs a gagné la partie. Dans ce cas, un message de victoire est envoyé au joueur gagnant et un message de défaite au perdant. Les deux interfaces se figent, les joueurs ne peuvent plus poser de pion sur la grille.