République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique.

Mémoire pour l'obtention
du diplôme d'ingénieur d'état en informatique.
Option : I.A

Sujet :

Etude Pour la Réalisation d'un multijeu intelligent sous J2ME

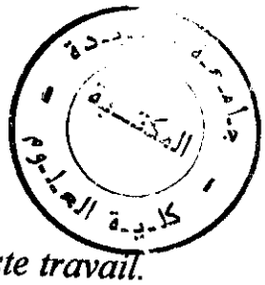
Présenté par : Korteby Farouk **Promoteur : Dr.Mahieddine Mohamed**

Organisme d'accueil : LDRSI.

Soutenue le:18/07/2006, devant le jury composé de :
Président:Mr.Nedjmi
Examinateur:Mr.Hantabli
Examinateur:Mr.Mazari

- 2005/2006 -

Remerciements



Avant tout je remercie « Allah » qui m'a aidé à réaliser ce modeste travail.

Je tiens à remercier mon promoteur M'. MAHIEDDINE Mohammed pour son aide, sa patience, sa disponibilité, et sa compréhensibilité.

Je remercie les membres du jury pour m'avoir fait l'honneur de juger mon travail.

J'adresse mes sincères remerciements à M^me^ Docteur OUKID pour nous avoir fourni un bon environnement de travail.

Je tiens aussi à exprimer ma profonde gratitude et mes vifs remerciements à toute ma famille surtout mes parents pour leur soutien durant toute ma scolarité.

Je remercie tous les enseignants de la faculté des sciences de BLIDA et surtout mes enseignants du département informatique.

Que tout le personnel du département d'informatique trouve ici ma reconnaissance pour son entière disponibilité.

Je remercie, de tout coeur, tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Dédicace

Je dédie ce modeste travail :

A mes très chers ***parents*** *pour leur soutien durant*
Toute ma carrière,
Pour leur bienveillance, leurs efforts constants dans mes études, et
Pour leur encouragements,

A mon frere et ma soeure

A tous mes amis hichem, readh, heythem, schahine, Amine, redha, mohamed, mounir, hocine et à tout ceux qui m'aiment.

Sommaire

Liste des Figures

Introduction

Introduction

Depuis sa création le marché des jeux vidéo est toujours en expansion. Car, après s'être développés sur les PC, les consoles de salons et les consoles portables, les jeux arrivent sur le marché mobile. La croissance de ce nouveau segment paraît presqu'aussi prometteuse que sur les PC, en voici quelques chiffres:

-chiffre d'affaire des ventes de jeux vidéo:

- En 2001 : 28 milliards de dollars (les jeux sur Mobile 0.7 million de dollars)
- En 2006 : 30 milliards de dollars (les jeux sur Mobile 3.6 million de dollars)

-le Nombre des joueurs de jeux sur mobile:

- En 2002 : 7 millions de joueurs
- En 2005 : 130 millions de joueurs
  (Source: [30] [31])

Les jeux mobiles devraient conduire la croissance mondiale des revenus du divertissement mobile. Selon les experts, les jeux mobiles deviendraient en 2009 la seconde source de revenus data, devant la musique. La figure qui suit illustre en million d'euros les dépenses totales des jeux mobiles sur les principaux marchés européens:

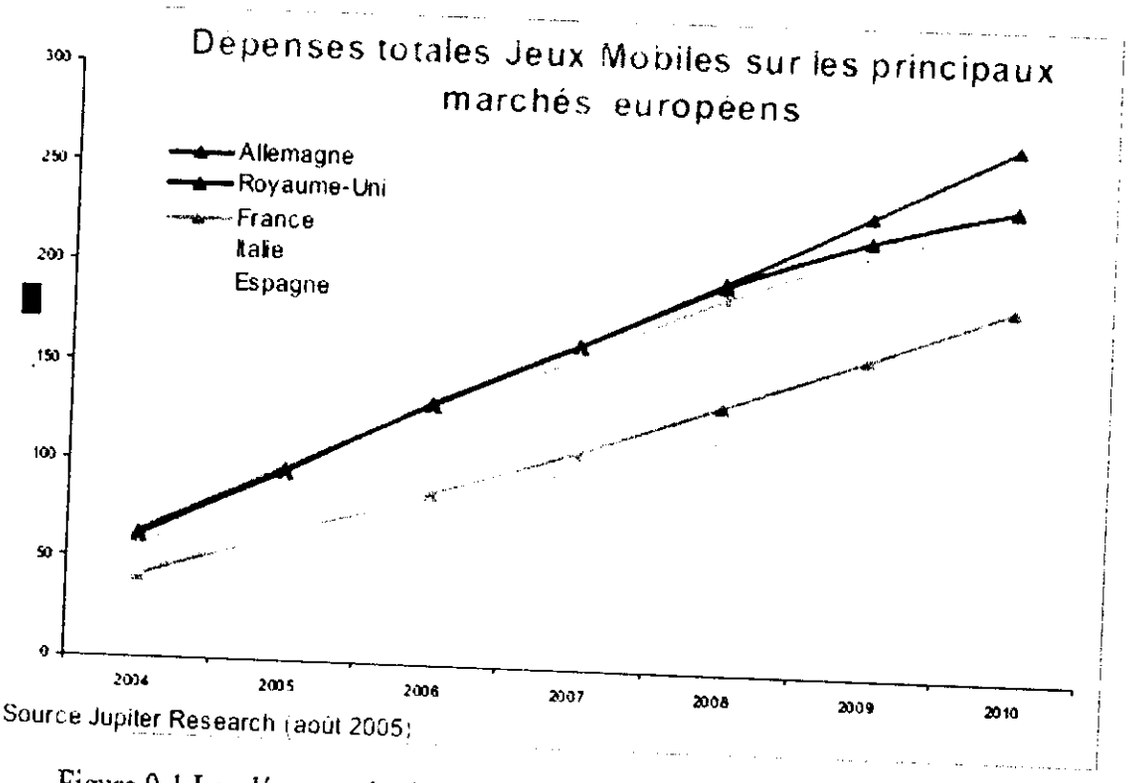


Figure 0-1:Les dépenses des jeux mobiles sur les principaux marchés européens.

Les joueurs mobiles : une population unisexe .En 2005, on estime à 130 millions le nombre de joueurs mobiles dans le monde. Différentes études menées en 2005 vont à l'encontre d'une idée reçue en démontrant que les joueurs constituent en fait une population unisexe.
Une étude menée par **I-Play**(une importante entreprise de création de jeux mobile) auprès de 2 500 clients mobiles montre, par ailleurs, que les femmes sont autant intéressées que les hommes par les jeux mobiles : 48% des femmes contre 44% des hommes.
Il y a trois principales catégories de joueurs mobiles peuvent être distinguées, dont les frontières ne sont pas toujours clairement définies :

1- Les joueurs occasionnels (**casual gamers** en anglais) jouent de manière fractionnée et sont stimulés par l'aspect communautaire des jeux et les compétitions. Les femmes représentent plus de 50% de cette population qui apprécient les jeux à ergonomie simple et facile à pénétrer (puzzle, jeux de cartes) : « Je clique, je joue ». Ils aiment les nouveautés et changent régulièrement de jeu.

2- Les joueurs réguliers sont généralement fidèles à un titre spécifique, ils sont fréquemment mono joueur mais l'aspect communautaire peut se révéler important. Ils constituent une population mixte et sont en moyenne âgés de 25 à 35 ans. Leurs titres favoris sont les jeux de gestion (casse tête) et les jeux de courses .Il sont généralement portés depuis l'univers des jeux PC.

3- Les joueurs intensifs sont principalement de jeunes hommes âgés de moins de 35 ans, qui sont attirés par la qualité graphique, les jeux d'action et de stratégie.

Voici la figure qui segmente les joueurs mobiles sur les années 2005-2010:

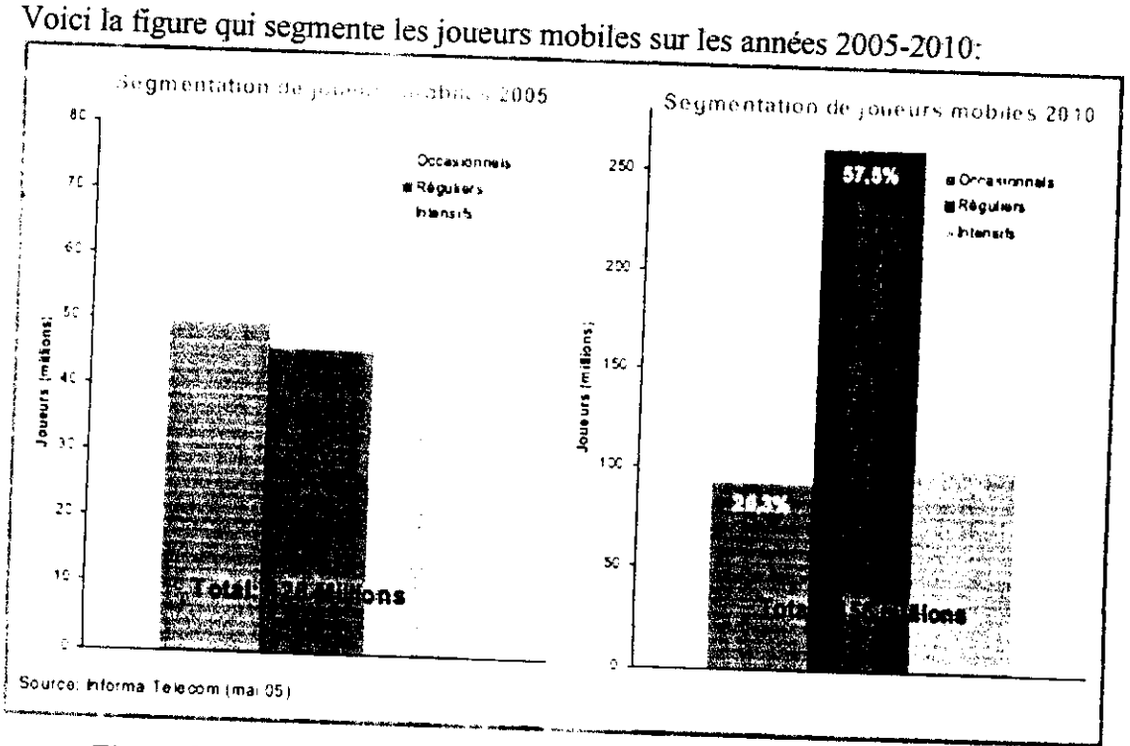


Figure 0-2:La segmentation les joueurs mobiles sur les années 2005-2010

On constate clairement l'augmentation très intéressante de la deuxième catégorie (les joueurs réguliers), Car ils vont constituer la plus grande part du marché (plus de 50%), portés depuis l'univers des jeux PC, ils cherchent toujours de nouvelles expériences.

Notre idée est de créer une nouvelle expérience de divertissement, jeux intelligents, dans les environnements mobiles pour les joueurs (réguliers), après que l'intelligence artificielle (spécialement le résonnement à base de règles) ait démontrée ses avantages dans les jeux PC.

Parmi les contraintes des environnements mobiles, on peut citer, leurs limites en matière de puissance de calcul, ce qui impose l'utilisation d'environnent de développement adapté a ces contraintes. Ces dernières années J2ME (Java 2 Micro Edition) s'est imposé comme l'environnement leader pour le développement sur mobiles.

Pourquoi J2ME?

Java 2 Micro Edition est une architecture technique dont le but est de fournir une solution de développement aux applications embarquées. L'intérêt étant de proposer toute la puissance d'un langage tel que Java associé aux services proposés par une version bridée du Framework J2SE (Java 2 Standard Edition): J2ME. Voici aussi quelques points de plus :

- **Java** est disponible sur la très grande majorité des téléphones mobiles anciens ou récents
- les Jeux et les programmes en **Java** offrent plus de performances que le **wap** ou les **sms** (graphismes, interactivité, ...)
- une fois qu'un programme est téléchargé, il peut utiliser gratuitement autant de fois que souhaité sans avoir nécessairement besoin d'être connecté au réseau.

Démarche suivie

Pour aborder notre problème, nous avons adopté plusieurs étapes selon la démarche suivante :

- Etude de la Plate forme J2ME ses avantages et ses différentes limitations.
- Etude conceptuelle de la plupart des jeux sur mobiles et sur pc existant.
- Faire une synthèse complète sur l'application de l'intelligence artificielle dans le domaine des jeux.
- Explorer le monde des systèmes experts dans le domaine des jeux.
- Etude et comparaison entre les différents moyens de communication dans un terminal mobile.
- Etude de la librairie jsr-82 qui permet d'utiliser la technologie Bluetooth avec J2ME dans les téléphones portables.

Après chaque étape, nous avons opéré par phases successives afin de vérifier la faisabilité de chacun des mécanismes étudiés. Ensuite, on a décidé d'introduire

l'intelligence artificielle, spécialement les systèmes experts, dans le développement des jeux sur terminaux mobiles, Ce qui nous permettra d'explorer un domaine carrément nouveau.

Plan du rapport

Notre mémoire est décomposé en cinq chapitres:

Chapitre 1: Jeux & Multijeux
Chapitre 2: IA & Théorie des Jeux
Chapitre 3: La Java 2 Micro Edition
Chapitre 4: J2ME & Réseau sans Fils (Bluetooth)
Chapitre 5: Conception & Implémentation

Nous présentons, dans un premier temps, les principales catégories de jeux existant avec quelques exemples. Puis, nous décrivons les principes des mécanismes de l'intelligence artificielle appliquée au domaine des jeux. Pour cela, nous présentons ses différentes formes notamment les systèmes experts. Suite à cela, nous présentons les divers composants de la plate forme J2ME .Nous décrivons, ensuite, les différents mécanismes de communications sans fil avec Bluetooth dans la plate forme J2ME. Avant de finir, Nous présenterons l'architecture de notre jeu, en décrivons sa conception et son implémentation.

Nous concluons ce rapport en rappelant les principaux résultats obtenus et dégageons quelques perspectives ouvertes pour un futur travail.

Chapitre 1 :

Jeux & Multijeux

1. Les Jeux & Multijeux:

1-1.Définition Jeu:

On peut définir le **jeu** comme une activité de loisirs d'ordre physique ou psychique, soumise à des règles conventionnelles, à laquelle on s'adonne pour se divertir tirer du plaisir et de l'amusement.

Le jeu, à l'instar du rire, est candidat au statut de **propre de l'homme** ; en effet, l'homme serait la seule espèce à jouer à l'âge adulte. Roger Caillois *dans « Les jeux et les hommes »* (Gallimard, 1967), s'est essayé à une définition du jeu. C'est une activité qui doit être :

1. *libre* : l'activité doit être choisie pour conserver son caractère ludique
2. *séparée* : circonscrite dans les limites d'espace et de temps
3. *incertaine* : l'issue n'est pas connue à l'avance
4. *improductive*
5. *réglée* : elle est soumise à des règles qui suspendent les lois ordinaires
6. *fictive* : accompagnée d'une conscience fictive de la réalité seconde

Pratiquement, toute activité humaine peut être l'objet d'un jeu, et réciproquement tout jeu peut cesser de le devenir (selon la juste expression enfantine : «*Sè pu du jeu !*»).

Cette dernière expression montre que le jeu est avant tout une institution (comme l'école, l'Assemblée nationale...), limitée dans le temps et limitée aux joueurs de la partie. Le jeu institue un espace de liberté au sein d'une légalité particulière définie par la règle du jeu (Colas Duflo, *Jouer et philosopher*, Presses universitaires de France, 1997).

Cependant, les deux définitions ci-dessus ne considèrent le jeu qu'en lui-même et pas dans ses rapports avec le monde réel extérieur au jeu. Le jeu est aussi une manière de représenter le monde. Ainsi le jeu transpose dans un objet concret des systèmes de valeurs ou des systèmes formels abstraits. De ce point de vue le jeu peut être considéré comme une métaphore du monde (ou d'une de ses parties). Jouer et/ou inventer un jeu, construire une partie en interaction avec son adversaire relève alors d'une activité culturelle de haut niveau, et chaque partie jouée est une forme d'œuvre d'art. [21]

1-2. Jeu et défi.

Le jeu n'est pas toujours compétition, comme le montrent les jeux de construction, ou le bilboquet. On imagine cependant mal un jeu de hasard ou d'adresse sans incertitude. Il semble pourtant que les jeux de rôles obéissent à un autre principe: participer d'une existence qui nous est inaccessible. Cependant, même dans ce cas, il y a bien une sorte de défi car il n'est pas si facile d'agir continuellement comme un autre. Selon Caillois, la règle du jeu est alors unique; elle consiste «à fasciner le spectateur, en évitant qu'une faute conduise celui-ci à refuser l'illusion». On peut de fait remarquer que dans certains jeux, il suffit, pour l'emporter, de respecter la règle le plus longtemps possible, de demeurer dans le jeu: ainsi le "ni oui ni non". Enfin, il y a une certaine proximité entre triompher du hasard, ou d'un adversaire, parfois de la

mort, mais dans un univers strictement conventionnel, et s'affranchir par l'imitation de ses propres limites. Caillois reconnaît encore une quatrième sorte de jeux, à côté des compétitions (agôn), des jeux de hasards (alea), et des jeux de rôles (mimicry). Il s'agit de ces activités qui n'ont pas d'autre but que le vertige (ilinx), comme de nombreuses attractions de fêtes foraines. Indiscutablement, elles enveloppent une dimension de défi, c'est-à-dire ici de courage physique. Quels que soient les enjeux, le beau joueur ne doit pas accorder trop d'importance à la victoire, ou à la défaite, parce que ce n'est précisément qu'un jeu, réputé sans conséquences. Pourtant, s'il ne leur accordait aucune importance, le jeu perdrait tout intérêt. Tout le déroulement de la partie d'échec se rattache à la préservation du roi, à son assimilation provisoire au Moi du joueur. Celui-ci doit être capable d'investir le plus intensément possible cette convention, et de retirer instantanément cet investissement lorsque la partie est finie. Le jeu est ainsi une création dont le joueur reste maître (Karl Groos). [21]

1-3.Types de Jeux

1-3-1-Jeu mathématique

Les jeux mathématiques incluent de nombreux sujets qui font partie des récréations mathématiques. Ce qui distingue un jeu mathématique d'un autre jeu ordinaire, c'est l'accent mis sur l'analyse mathématique du jeu, la logique nécessaire à son accomplissement, plus que sur la façon de jouer.

En Anglais, "Mathematical Games" était le titre d'une rubrique de Martin Gardner dans la revue Scientific American, ayant passionné de nombreuses générations de mathématiciens et de scientifiques. Douglas Hofstadter et Ian Stewart ont successivement pris le relais à travers les rubriques Metamagical Themas et Mathematical Recreations. [21]

Exemples de jeux mathématiques et autres casse-tête

Il existe une Multitude de type de jeux, qui se diversifient en terme de nature et de technique de jeux, parmi ces jeux on peut citer [21] :

Jeux logiques et casse-tête

- les tours de Hanoi
- le rubik's Cube

Crucinumérisme

- les nombres fléchés
- le sudoku

Jeux de pavage

- le pavage de Penrose
- le tangram

Jeux combinatoires abstraits résolus

- les jeux de Nim
- Le jeu de Hex

Récréations mathématiques

- le jeu de la vie
- l'Origami
- les Sangaku sont des tablettes mathématiques votives visibles dans certains temples japonais et figurant des énigmes géométriques. Ces objets établissent un lien avec la vie artistique et la vie religieuse par le biais des mathématiques. [21]

1-3-2. Jeu linguistique

- Jeu de mots
- Mots croisés (et variantes)
- Scrabble
- Boggle

1-3-3-Jeu de société

On trouve dans cette famille de très nombreuses variétés de jeux : jeux de réflexion, jeux de hasard pur ou raisonné, jeux de déduction, jeux de lettres, jeux de connaissance, etc.

On se reportera à l'article Jeu de société pour les différents types de jeux de société. Certains jeux ont créé des genres devenus tellement importants, avec des communautés de joueurs qui s'y consacrent presque exclusivement. Ils constituent ainsi des familles de jeux à part entière même s'ils sont, fondamentalement, des jeux de société.

C'est par exemple le cas des jeux de simulation, dont les règles visent à décrire de manière fine certaines situations : [21]

- Jeu de rôle
- Jeu de figurines
- Jeu de guerre

1-3-4-Jeu solitaire

- Casse-tête
  - Cube de Rubik
  - Énigme
  - Sudoku
- Puzzle
  - Tangram
- Patiences

1-3-5-Jeu par correspondance

Jeux par courrier postal, puis plus récemment par courriel électronique ou encore par navigateur Internet, gratuits le plus souvent. Ils sont parfois désignés par l'abréviation anglaise PBeM (Play by e-mail).

Ils peuvent être sous plusieurs formes :

- La mailing list, chacun effectue son tour par l'envoi d'un mail contenant les ordres de jeu.
- Le forum, où l'on pourra directement lire les actions entreprises par les autres joueurs, et passer les siennes.
- Le site, qui se passe de toute intelligence humaine et permet de constater immédiatement le résultat de ses actions.

Ces jeux ont pris leur essor avec l'Internet, les comptes mail gratuits et grâce aux hébergeurs de sites web gratuits. [21]

- L'Archipel du Micromonde
- Élevage d'animaux virtuels sur Internet

1-3-6-Jeu de plein-air [21]

- Agrès : tourniquet, toboggan, balançoire...
- Jeu de plage et sports : beach-volley, jokari, frisbee
- Jeu de piste
- Jeu d'équipes : 1, 2, 3 soleil, Marroneur, Zagamore
- Jeu de rôle grandeur nature (ou « semi réel »)
- Jeu de bassin : pataugeoire, bac à sable (jeu d'enfant)

1-3-7-Jeu à monnayeur

On peut regrouper sous cette dénomination tous les jeux auxquels il est possible de jouer dans un café ou une salle de jeux. Insérer une pièce dans le monnayeur permet d'acheter une ou plusieurs parties des jeux suivants [21] :

- le flipper
- le billard
- les fléchettes
- les jeux d'arcade
- le baby-foot
- le jeu de palets

1-3-8-Jeu vidéo

1-3-8-1. Définition

Les jeux vidéo utilisent des moyens techniques spécifiques, soit des consoles de jeu vidéo, des bornes d'arcade, des ordinateurs ou encore des appareils Mobiles. D'invention récente (dans les années 1980), ils ont ouvert le champ à de nouvelles manières de jouer, plusieurs types de jeu vidéo ont vu le jour. Voici une liste non exhaustive[21] :

- le jeu de simulation
- le jeu de plates-formes
- le jeu de tir subjectif (*FPS*)
- le jeu d'aventure
- le jeu de stratégie en temps réel (*RTS*)
- le shoot them up
- le tamagotchi (animal de compagnie virtuel)
- le jeu de rôle (*RPG*)
- le jeu en réseau
- le jeu en ligne massivement multijoueur (*MMORPG*)
- le jeu sur Internet (*site jeu*)

1-3-8-1. Histoire du jeu vidéo

Le début de l'histoire du jeu vidéo est relativement flou compte tenu du fait que la notion même de jeu vidéo n'est pas précisément cadrée et constitue l'objet de débats entre spécialistes. Donc selon la définition que l'on accepte du jeu vidéo, son histoire peut commencer aux alentours de 1950 avec l'idée de Ralph Baer ou bien 1952 avec *OXO*, 1958 avec *Tennis for Two* ou encore 1962 avec *Spacewar*, qui est la date la plus communément admise. *Pong* quant à lui est le premier jeu dont le *gameplay* a été suffisamment accrocheur et addictif pour lui faire connaître le succès auprès du grand public.

Si *Pong* n'a pas inventé le jeu vidéo, il est incontestablement le jeu ayant donné le coup d'envoi à l'industrie vidéo ludique. Celle-ci connaît une croissance explosive et fébrile aux États-Unis jusqu'en 1983 où elle subit un krash qui la fait migrer vers le Japon. C'est là qu'elle voit sa renaissance, notamment grâce à la NES de Nintendo et au jeu *Super Mario Bros* en 1985 qui inaugure une nouvelle philosophie dans la conception des jeux vidéo : plus riches et ouverts à tous les publics. Depuis, le jeu vidéo est en croissance continue. [21]

1-3-8-1. Différentes plates-formes de jeu

Il existe quatre grands types de matériel sur lesquels il est possible de jouer à un jeu vidéo. Surtout depuis l'avènement des consoles 128 bits, de nombreux de jeux sont disponibles sur les quatre environnements dominants :

a- Jeu sur ordinateur

Les jeux vidéo sur ordinateur sont des logiciels qui fonctionnent sur un ordinateur personnel équipé d'une interface de jeu. Parmi les nombreuses interfaces existantes, on peut citer le fameux duo clavier/souris qui est spécifique à l'ordinateur et permet une grande précision grâce à la souris et la possibilité d'accéder rapidement à de nombreuses commandes grâce aux raccourcis claviers. Cependant, pour un meilleur confort, le joueur peut brancher d'autres périphériques qui lui apporteront un meilleur confort pour des jeux spécifiques. Ainsi, il est possible de brancher un *joystick* pour les simulations de vol, un volant pour les jeux de courses ou encore une manette de jeu (à l'instar du périphérique principalement utilisé par les consoles de jeu modernes). L'affichage s'effectue sur l'écran de l'ordinateur, éventuellement au moyen d'une sortie vidéo (pour affichage sur grand écran, par exemple) ou casque 3D. Le rendu sonore du jeu est retransmis via des haut-parleurs externes, ou une sortie audio vers un dispositif d'amplification externe (chaîne Hi-Fi, par exemple).

Tous types d'interactions Homme/machine sont envisageables, dans la mesure où il est relativement facile de modifier le logiciel pour le rendre utilisable avec un matériel quelconque [21] .

b- Jeu sur console

Les jeux sur console sont des jeux prévus pour fonctionner sur un matériel entièrement dédié à ce loisir, la « console de jeux vidéo ». Initialement un matériel particulier, muni de contrôleurs dédiés à la production de son, d'images et de périphériques d'entrée, la console de jeu tend de plus en plus à ressembler à un ordinateur personnel, avec des interfaces simplifiées et vouées aux contrôleurs de jeu. On branche la console de jeu sur un téléviseur, ou sur un vidéo projecteur. Elle peut être munie d'une sortie audio destinée à être branchée sur un amplificateur de chaîne Hi-Fi. Il existe aussi des consoles de jeu autonomes, appelées consoles portables. Ces consoles intègrent tout le matériel dont a besoin une console pour fonctionner (écran, contrôleur, son) dans une unité nomade, alimentée par des piles ou une batterie rechargeable.

Sur console, le joueur interagit avec une manette de jeu ou *joypad* , par opposition au *joystick*, mais aussi par l'intermédiaire de périphériques variés pouvant aller du fusil optique ou laser, au volant à retour de force.

Il existe des consoles dédiées uniquement à un jeu, mais l'écrasante majorité des consoles fonctionne avec des jeux interchangeables. Ceux-ci se présentent sous forme de cartouches ou de tout autre support numérique (CD ou DVD par exemple) [21] .

c- Jeu sur borne d'arcade

Les jeux d'arcade sont une catégorie particulière de jeux vidéo, étroitement liée au matériel. Une borne d'arcade se compose classiquement d'un monnayeur, d'un contrôleur de type manette de jeu et d'un certain nombre de boutons poussoirs. On peut trouver toute une gamme de périphériques divers : fusil optique, volant à retour de force, siège inclinable, etc... On distingue deux grandes catégories de bornes d'arcade, les bornes dédiées et les bornes génériques. Les bornes dédiées ne

contiennent qu'un seul jeu (indissociable de ladite borne), et sont généralement les premières bornes d'arcade ayant existé, ou celles nécessitant des périphériques spécifiques à un jeu donné. Les bornes génériques sont, quant à elles, apparues au début des années 1980 grâce à une standardisation qui a conduit à la création du connecteur JAMMA permettant de changer le jeu à l'intérieur de la borne en seulement quelques minutes. On citera aussi le célèbre format MVS (équivalent arcade de la Neo Geo de SNK) [21].

d- Jeu sur Mobile

La plupart des téléphones portables maintenant sont livrés avec un ou plusieurs petits jeux. Tétris, démineur, solitaire... Pas très évolués, ils sont en outre plutôt simplistes graphiquement. Il est possible d'en ajouter de bien plus amusants : courses de voitures, jeux de stratégie, de plates-formes, de sports, etc. On trouve même des adaptations de grands classiques sur PC[09].

En effet, grâce à la technologie **Java,** compatible avec de nombreux mobiles, il est possible d'obtenir des jeux qui exploiteront bien mieux les possibilités des téléphones et des écrans couleur qui équipent les modèles récents alors plusieurs types de jeu vidéo ont vu le jour parmi eux [05] :

d-1. Les Jeux de SMS

Le **Simple Messaging System** est une méthode simple d'envoyer des messages a travers des terminaux mobiles alors la seul façon de représentes le jeu est d'utiliser le texte comme interface ou bien graphiques, il y a beaucoup de jeux de SMS mais ils sont tous très simple : vous envoyez un message a un numéro particulier et vous recevez, en contre partie un autre message du serveur du provider de ce jeu.
On peut cites quelques exemples des jeu de SMS [05]:
Fisupeli : jeu de pêche (Finlande)
BotFighters by It's Alive : jeu de combat (Suède)

d-2. Les jeux du WAP

Le Wireless Application Protocol (WAP),c'est une très simple plateforme dans laquelle tous les jeux doivent être télécharger dans le mobile comme des petites pages Web,vous pouvez lire du texte ou bien voir des graphiques en noir et blanc,et entrer des informations dans des Formes mais le transfère de donnes est malheureusement très lent et demande des minutes pour affiches ses jeux [05].
On peut citer quelques exemples comme:

Sorcery
C'est un jeu d'aventure, vous étés en quête de chercher la demeure des rois (Crown of Kings), résolvez des puzzles, Combattre des monstres et concevoir votre joueur


Figure 1-1:Sorcery

WAP Massacre
C'est un jeu de combat qui vous procure une sensation de frisson.

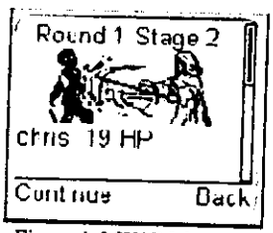


Figure 1-2:WAP Massacre

d-3. Les jeux du i-mode

La plateforme i-mode est similaire au WAP mais en général elle est plus rapide et offre une très grande capacité d'affichages des couleurs et elle comporte une centaines de jeu on vas citer quelques exemples [05]:

Dwango's Turibaka Kibun (la pêche fou en français)

Jeu de pêche japonais très connus au japon car sa compagnie a compté plus de 1.5 millions participants dans son jeu.


Figure 1-3:la pêche fou

d-4. Jeux sur iAppli

NTT DOCOMO'S une version d'applet java mais sur téléphone portable c'est très populaire en japon appelées aussi iAppli, cette version supporte le son, toutes les couleurs et une grande variété de graphiques et animations .il y a beaucoup de sociétés qui ont investie dans cette technologie parmi elle Disney Internet Group International (DIG) devenus très vite le leader dans ce domaine [05]. Voici quelques exemples :

Dwango's Samurai Romanesque

Un jeu de combat, il coûte 3$ /mois et supporte le mode multi joueurs, un serveur peu contenir jusqu'à 500 000 joueurs


Figure 1-4:Dwango's Samurai

Simultanément et ils peuvent faire des concessions, se battre, faire du commerce et bavarder les uns avec les autres.

The Billionaire: Un jeu de cartes très populaire


Figure 1-5:Billionaire

Chess: un Jeu d'échec

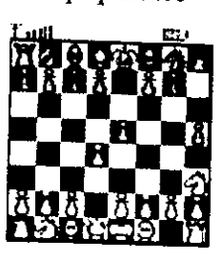


Figure 1-6:Chess

d-5. Les Jeux J2ME MIDP

J2ME est le framework Java spécialisé dans les applications mobiles. Des plates-formes Java compatibles avec J2ME sont embarquées dans de nombreux téléphones portables et PDA [08].

Une plate-forme J2ME est composée :

- d'une KVM (Kilobyte Virtual Machine), une machine virtuelle capable d'exécuter une application Java
- d'une « configuration », une API donnant accès aux fonctions de base du système
- d'un « profil », une API donnant accès aux fonctions spécifiques de la plate forme.

Les configurations les plus courantes sont [08]:

- CLDC (Connected Limited Device Configuration), que l'on retrouve par exemple dans les téléphones mobiles
- CDC (Connected Device Configuration), qui est plutôt utilisé dans des décodeurs de télévision numérique

On va plus détailler cette plate-forme dans les chapitres a venir et maintenant on voir quelques exemples de jeu:

Karl Hörnell's MIDP-Man

C'est la version mobile du célèbre jeu Pac-Man

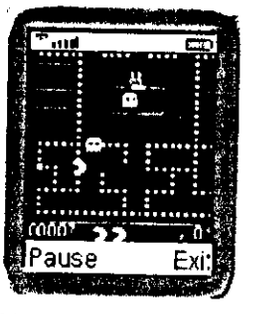


Figure 1-7:Pac-Man

Super Pool

Un jeu multi joueurs de billards implémentant la technologie Bluetooth.

Figure 1-8:Super Pool

Axion:
C'est un jeu d'arcade, avec un vaisseau vous survolez plusieurs paysages et éviter tout genre de vaisseaux ennemis. Avec votre progression dans le jeu vous recevrez différentes armes et type de missiles.

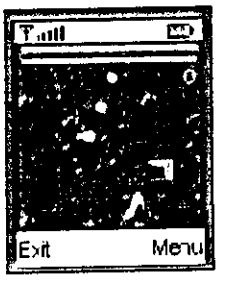


Figure 1-9:Axion

Spruce Driver:

C'est un jeu de course (Racing Game) très rigolo avec des obstacles.

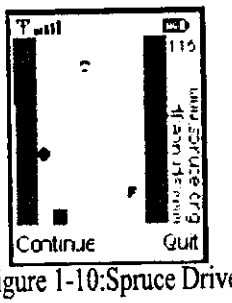


Figure 1-10:Spruce Driver

1-4. Introduction et Historiques des jeux Multi Joueurs:

Les jeux multijoueurs ont existé depuis très longtemps. Les premiers exemples peuvent être localisés au début des années 1980, avec l'adoption massive d'Internet et du Web avant les années 1990. Ils peuvent offrir les mêmes ressources multimédias de jeux au joueur simple mais en introduisant d'autres joueurs. Cela rend les défi plus grand et, à l'opinion de beaucoup de gens, plus il y a de joueurs plus en vaut la peine de jouer [06].

La raison pour laquelle les jeux de multijoueur ont été si réussis grâce a la présence de l'intelligence des adversaire humain. Plus la partie sociale non existante dans les jeux à joueur simple.

Un des premiers exemples réalisables d'un jeu de multijoueur était Essex MUD, qui a été développée par **Richard Bartle** et **Roy Trubshaw** d'Université D'essex en 1979. Le jeu original s'est composé d'environ 20 pièces décrites dans leurs discours. Jusqu'à 250 joueurs pourraient coexister dans ce monde de jeu, donc ils pourraient "se voir" dans la pièce. Les textes descriptifs déclareraient des événements comme "Peter est ici,". par exemple Les joueurs connecte au serveur de MUD utilisant le réseau, qui est raccordé aux six universités britanniques; et à partir de 1980, les joueurs connecte au serveur de MUD utilisant le réseau ARPA des États-Unis. Comparé avec un jeu moderne, **Planetside** (par Sony) qui supporte plus de 3,500 joueurs par serveur.

Le Essex MUD avait inspiré beaucoup de développeurs. En fait, une scène de Essex MUD a existé pendant les années 1980 et une partie des années 1990.les Essex MUD se sont éteintes avec des jeux de multi joueurs comme **Everquest** . Mais avant 10 ans, ils ont gouverné l'arène de multi joueurs [06].

Pourtant, les Essex MUD n'étaient pas le seul jeu .en devenant ainsi une expérience de jeu réalisable, ils y'avait d'autres jeux compétitifs simples avec des adversaires humains avec plus de défis, Un bon exemple est M.U.L.E, jusqu'à quatre joueurs ont rivalisé pour conquérir une nouvelle planète dans une galaxie très lointaine, des jeux de stratégie en temps réel .Mais leurs succès a été limité, surtout parce que l'infrastructure de réseau en 1983 n'étais pas prête pour les jeux a temps réel.

Le tournant a repris autour de l'année 1993, quand Internet et plus spécialement le phénomène de Web fait exploser. Dans moins de six mois, le Web évoluait du zéro à l'infini, en prenant la planète entière par sa tempête. Comme un effet indésirable, les vitesses de connexion ont été beaucoup améliorées, du très premier 9600bps les modems à quelque part autour de 56kbps et de là a ISDN, DSL et le câble. Cette augmentation de vitesse était extrêmement pertinente parce qu'il a permis aux développeurs de transférer tous les renseignements publics d'un ordinateur à un autre en gardant le jeu tourne. par exemple, L'âge d'Empires, pourrait être joué sur un modem 56kbps et cela a inclus des douzaines de joueurs luttant l'un contre l'autre.

Aujourd'hui, les jeux de multijoueur ont stabilisé leurs stratégies sur environ deux "orbites", qui sont les descendants directs de Essex MUD et M.U.L.E.

Asheron's Call et beaucoup d'autres jeux. Sont tous base sur le fait de communiquer en société avec un monde de jeu très riche. Des milliers des gens vivent vraiment leurs vies à l'intérieur de chacun de ces jeux dans une réalité alternante. En fait, ces jeux sont probablement les plus additifs d'eux tous, avec les cas extrêmes comptant plus de 1,600 heures de jeu par an (c'est plus de quatre heures par jour, même les week-ends). Comme un phénomène, les économies virtuelles réelles ont été créées. Certains fans ont vendu leurs propriétés ou caractères sur les sites Internet de vente aux enchères, souvent pour les sommes d'argent respectables. Ce type de conduite devrait sans doute nous faire penser à l'influence de la scène de jeux sur les vies des gens.

En fin , les jeux multijoueur a échelle réduit ont réussis aussi a nous faire une distinction entre les jeux multijoueur compétitifs et coopératifs. La différence est dans le rôle des autres joueurs : seront-ils des ennemis que vous devez vaincre, ou seront-ils vos coéquipiers sur une mission globale ? Une distinction qui a fait la popularité incroyable du jeu **Counterstrike** , ou bien sont ancêtre **Half-Life** . Dans ses jeux, deux équipes luttent l'un contre l'autre, ainsi il y a une partie compétitive par rapport les joueurs opposés et une autre partie coopérative entre les membres du même côté.

L'avenir semble brillant pour les jeux de multijoueur. L'infrastructure de réseau continue à grandir et aujourd'hui la plupart des utilisateurs ont déjà une connexion Internet de large bande qui permet transmission a haut débit. Plusieurs compagnies de création de jeux en étais crée au cours des dernières années, en développent de plus en plus de meilleurs jeux avec plus d'innovations. Certaines personnes prédisent que les jeux de multijoueur engloutiront finalement tous les jeux de joueur simple. Bien que ce soit probablement une exagération, elles montrent clairement le niveau d'attentes l'industrie par rapport les jeux multijoueur. Mais souvenez-vous les sports en solo et les sports d'équipe coexistent paisiblement dans le monde réel, ainsi il n'y a vraiment aucune probabilité que les jeux de joueur simple vont disparaître en faveur des jeux multi joueurs [06].

1-5. Conclusion

Après avoir vue une brève introduction aux jeux et leurs différents types notamment les jeux sur Mobile .On va maintenant présenter l'intelligence artificielle appliquer à la théorie des jeux, cela pour rendre les jeux plus intéressants.

Chapitre 2 :

IA & Théorie des Jeux

2- L'IA & la Théorie des Jeux :

2-1. Introduction a L'IA

2-1-1. Définition de L'IA

L'**intelligence artificielle** (terme créé par John McCarthy), souvent abrégée avec le sigle **IA**, est définie par l'un de ses créateurs, Marvin Lee Minsky, comme « *la construction de programmes informatiques qui s'adonnent à des tâches qui sont, pour l'instant, accomplies de façon plus satisfaisante par des êtres humains car elles demandent des processus mentaux de haut niveau tels que : l'apprentissage perceptuel, l'organisation de la mémoire et le raisonnement critique* ».

Depuis la nuit des temps, l'imaginaire collectif semble peuplé de créatures intelligentes créées de toutes pièce par l'être humain (le golem juif, la statue de pygmalion...). Ces contes semblent traduire un désir profond de l'homme de reproduire les mécanismes à la base de son intelligence. Outre ces visions fantasmatiques, l'histoire dénombre de multiples tentatives concrétisées, pour construire des machines capables de se substituer à l'homme pour effectuer à sa place certaines tâches ``intellectuelles'' répétitives ou fastidieuses (le boulier, la machine de Pascal, les automates de Vaucanson, etc.). L'apparition des ordinateurs a permis de cristalliser autour d'un ensemble de paradigmes, la compréhension et la reproduction des mécanismes de l'intelligence [04].

2-1-2.Historique de L'IA

L'IA est actuelle est issus d'un développement de l'aspect humain avec l'aspect machine en passant par plusieurs phases d'évolution dans les domaines suivants :

PHILOSOPHIE :

- 1596-1650 : Descartes suppose que les animaux sont des sortes de machines vivantes. Les hommes sont dotés d'une âme échappant à la matière.
- 1646-1716 : Leibniz introduit le matérialisme. Selon lui, tout est régit par des lois physiques.
- 1561-1626 : Bacon et l'empirisme : tout ce qui atteint la compréhension l'est par l'intermédiaire des sens
- 1711-1776 : Hume construit son principe d'induction : les règles générales sont acquises par exposition à des associations répétées de leurs éléments
- 1872-1970 : Russel formalise les travaux de Hume et introduit le positivisme logique : le savoir est caractérisé par des liens logiques, finalement reliés aux sens.

MATHS :

- 9ème siècle : al-Khowarazmi introduit l'algorithmique, l'algèbre et la notation arabe
- 1815-1864 : Boole construit l'algèbre binaire et la logique formel
- 1848-1925 : Frege établit la logique du premier ordre

- 1906-1978 : Gödel établit son théorème d'incomplétude et d'indécidabilité (il n'existe pas de procédure capable de décider qu'un non-théorème en est un).
- 1912-1954 : Turing montre que toute fonction calculable l'est par une machine de Turing (et donc par un ordinateur). Attention ! il existe des fonctions non-calculables.

PSYCHOLOGIE :

- 1821-1894 Helmoltz, 1832-1920 Wundt : origine de la psychologie scientifique
- 1878-1958 Watson, 1874-1949 Thorndike : introduction du behaviorism : le comportement d'un individu est le produit de conditionnements multiples.
- 1948 Tolman : Le cerveau contient des représentations internes sous forme de cartes cognitives

AUTOMATES :

- 1748 : Julien Offroy de la Mettrie assimile l'homme à une machine complexe
- 1709-1782 : Vaucanson construit des automates (canard, joueur de flute traversière) qui donnent l'illusion d'être vivants.
- 1592-1635 : Schickard créé la première machine à calculer à l'aide d'engrenages (addition, soustraction, multiplications, mémorisation de résultats et dépassement de capacité).
- 1623-1662 : Pascal (1642) réalise la ``Pascaline" ? Leibniz construit une machine effectuant les 4 opérations élémentaire.
- 1785-1870 : Thomas de Colmar construit ``l'arithmomètre"
- 1792-1871 : Babage réalise une ``machine analytique" programmable grâce à des cartes de variables et des cartes d'opérations (il est à noter que Ada de Lovelace, fille de Lord Byron, fut la première personne à concevoir des programmes. C'est en son honneur que le langage Ada porte son nom).

Il faut attendre les travaux de Turing (1912-1954) pour voir apparaître la première tentative délibérée pour comprendre et reproduire l'intelligence humaine. Turing construit son action autour du paradigme fondateur consistant à affirmer que ``toute l'intelligence cognitive humaine est modélisable dans le cadre formel de la machine de Turing". La machine de Turing est une machine abstraite capable d'effectuer des calculs. Elle est composée d'une unité de traitement et d'une mémoire qui n'est autre qu'un ruban à partir duquel la machine peut lire données et programme, stocker des informations temporaires et imprimer les résultats. Un ordinateur est donc une machine de Turing améliorée. Turing propose un test, capable, selon lui, de décelé si un système a reproduit l'intelligence humaine. Ce test consiste à faire dialoguer un humain et le système et déterminer si l'humain peut déceler si le système n'est pas humain[04].

INFORMATIQUE :

- 1940 : Heath Robinson crée par l'équipe de Turing pour décoder les messages allemands (technologie à base de relais)
- 1943 : Colossus, encore créé par l'équipe de Turing (technologie à base de lampes à vide)

- 1940-42 :
  - Z3; ordinateur programmable doté du premier langage évolué
  - ABC; Iowa state university
  - Mark I, II et III; Harvard
  - ENIAC : University of Pennsylvania
  - EDVAC : Programme mémorisable (sur les conseils de Von Neumann)
  - IBM701
  - 1965-1980 Mise au point d'algorithmes efficaces. Prise de ce conscience de la grande complexité du problème
  - 1982 ordinateurs de la 5eme génération (Japon) pour 1992 une machine parallèle capable de raisonner.

Avec l'apparition des machines massivement parallèles le niveau de représentation de l'information utilisant des ``symboles" est il toujours pertinent ?

INTELLIGENCE ARTIFICIELLE :

- 1943 : McCulloch et Pitts créent le modèle du neurone formel
- 1948 : Création de la cybernétique (science des systèmes) par Norbert Wiener.
- 1949 : Hebb établit la première règle d'apprentissage neuronal
- 1950 Shannon, 1952 Samuel, 1953 Turing : machine pour jouer aux échecs
- 1956 Workshop où est né le terme ``intelligence artificielle"
- 1959 Rochester : Geometry Theorem Prover
- 1958 McCarthy au MIT créé le LISP et le ``time sharing". Créé DIGITAL.
- 1960 John McCarthy, Allen Bewell & Herbert Simon: L'ordinateur peut être utilisé pour autre chose que des calculs ``manipuler des symboles" (idée proposée par Ada Lovelage amie de Babbage 1842)
- 1969 arrêt des RNs (Minsky & Paper 1969) limitations des perceptrons
- 1969-1979 : systèmes experts
- Depuis 1986 : retour des réseaux de neurones

Les langages d'IA :

- Lisp 1956/58 John McCarthy au MIT
- Les moteurs d'inférences: l'importance de l'organisation des données (structure, liste...) Programmer à partir des données
- Prolog 1973 Colmerauer : Générateurs de Systèmes Experts

Le développement de l'intelligence artificielle a été conditionné d'une part par le cloisonnement des sciences du vivant et des sciences pour l'ingénieur (l'IA se réclamant des sciences exactes), et d'autre part par une orientation principalement exclusivement basée sur le paradigme de Turing. C'est pourquoi, malgré des débuts ancrés dans la biologie et la psychologie, l'IA est rapidement passé à un formalisme mathématique très poussé et s'est attaquée à des problèmes plutôt symboliques. Elle s'est d'autant plus confortée dans ces idées que les débuts de la cybernétique (Widrow60 : adaline, Rosenblatt62 : perceptron), ont conduit à une impasse du fait de la démonstration de Minsky69 de la limitation des perceptrons [04].

2-1-3.Les principales applications de l'IA

Les principaux domaines d'application de l'IA sont :

- la traduction automatique (initialisé pendant la guerre)
- Les systèmes d'aide (au diagnostic, à la programmation)
- Les système de résolutions de problèmes -- GPS (general problem solver) A. Newell & H. Simon 1978
- Les jeux La simulation
- ELIZA J. Weizenbaum 1960 (simulation d'un dialogue avec un psy)

Cependant, cette approche s'est trouvée confrontée à de nombreux problèmes :

- Problème de la traduction automatique (Symbol Grounding Problem)
  - Problèmes de la traduction syntaxique:
  - anglais —russe —anglais
  - ``l'esprit est fort, mais la chair est faible" —``La vodka est forte, mais la viande est pourrie"
- Problème sur les applications à taille réelle
- Modélisation du sens commun
- Explosion combinatoire
- L'apprentissage
- Niveau de représentation
- Symbol Grounding Problem (ex. de la chambre chinoise).
- Complexité/chaos

2-1-4. Problématique de l'IA

Le but d'un système d'intelligence artificielle est d'aider à la résolution d'un problème posé par l'utilisateur. Mais qu'est-ce qu'un problèmes (comment décrire le problème, connaissances a priori) ? Quel est le niveau d'autonomie de l'IA (peut-il y avoir interaction entre l'utilisateur et le système et de quelle manière) ? Le système doit il gérer un problème particulier ou le spectre de résolution est il large (peut on généraliser le problème, quelle est l'influence du changement d'échelle, y'a-t'il plusieurs solutions, veut-on la meilleure solution ou une bonne solution) ? Quel est le degré d'évolutivité du système (peut-il assimiler de nouvelles données/apprendre) ?

Répondre à ces questions nous oriente vers un type de résolution de problèmes adapté (optimisation, recherche dans un espace d'états, logique, réseaux de neurones supervisés ou non). Toutefois, il est parfois encore possible de résoudre le problème de différentes manières. Le choix se fait alors en fonction de 2 types d'approches :

- Une approche analytique et formelle
- Une approche holiste essayant d'intégrer la complexité

Ces 2 approches sont souvent contradictoires car elle amène, pour la première, à décomposer le problème en partant d'une description abstraite puis en la raffinant petit à petit (approche ``top-down") alors que la deuxième tente plutôt de construire un système en intégrant hiérarchiquement des mécanismes de plus bas niveau (approche ``bottom-up") [04]

La décision finale est alors laissée à la discrétion du programmeur en fonction de ses applications, de ses habitudes, de ses goûts ou de ses convictions !

2-2. Les jeux Intelligents

2-2-1. Historique des jeux Intelligents

C'est à partir du XVII ème siècle que les premiers travaux sur les jeux ont été entamés, notamment avec Pascal et Fermat, qui ont commencé à étudier les probabilités. Au XVIII ème siècle, grâce aux travaux de BERNOUILLI sur le dénombrement, les combinaisons et les lois de
Probabilités, certains jeux ont pu être résolus. Grâce aux économistes Cournot et Edgeworth au début du XX ème siècle, certaines applications économiques ont été trouvées et modélisées (Ex : le Duopole de Cournot : situation de conflit entre deux entreprises qui sont soumises à des quotas).
La théorie moderne des jeux a vu le jour grâce à l'ouvrage désormais classique « **The Theory of Games and Economic Behaviour** », écrit par Von Neumann, publié en 1944 et qui jette alors les bases de la micro-économie. Néanmoins, le résultat le plus important de cette théorie fut fourni dès 1928 par le même homme : il s'agit du théorème du minmax, que l'on citera dans la partie consacrée aux algorithmes de jeux. Ensuite, dans les années 50, c'est surtout grâce aux travaux de Nash, qui aboutirent à la notion d'équilibre (de Nash), que la théorie
Progressa. Ces travaux remarquables valurent à Nash et à 2 autres chercheurs le prix Nobel d'économie en 1994. Depuis, la théorie des jeux a largement progressé notamment dans les jeux dits coopératifs et les jeux non-coopératifs répétés. Grâce aux travaux d'Axel Rod en 1984, on a découvert l'extraordinaire potentiel de l'étude de ces jeux répétés dans des applications non économiques.
Depuis le début des années 90, de nombreuses recherches sont entreprises, notamment sur les jeux répétés avec les simulations par automates finis (Ex : le programme baptisé « PRISON » de l'université de Lille I sur la coopération dans le dilemme du prisonnier). La théorie des jeux fait ainsi partie des domaines les plus prometteurs du XXI ème siècle [14].

2-2-2. Les différents types de jeux

2-2-2-1. Les différents types d'information

L'information dans un jeu peut être a informations complète, incomplète, parfaite ou imparfaite. Un jeu est dit à information *complète* si chacun des participants connaît :

- ses possibilités d'actions (l'ensemble des choix qu'il peut faire)
- l'ensemble des choix des autres joueurs
- les issues possibles et la valeur des gains qui en résultent
- les motifs des joueurs : chacun sait se mettre à la place des autres et sait ce que l'autre.

Déciderait s'il était dans la même situation. Cette hypothèse est la rationalité, toujours supposée de l'adversaire : tous les joueurs tentent de maximiser leurs gains et il n'y a que cela qui les intéresse.

On dit qu'un jeu est à information incomplète s'il manque de l'information (lorsqu'il n' y a pas de connaissance des gains, ou de certaines règles ...). Par la suite, on ne considèrera par défaut que tout les jeux sont information complète.

Dans les jeux à information complète, l'ordre des coups permet de distinguer 2 types de jeu:

- S'il y a *simultanéité des coups*, comme dans le jeu des enfants où l'on choisit simultanément ciseaux, pierre, feuille. On ne peut alors pas se décider en fonction de ce que joue l'adversaire puisqu'on joue en même temps. On dit qu'il y a une *information imparfaite.*
- Dans le cas du jeu d'échecs par contre, les coups n'étant pas simultanés, mais *successifs*, vous disposez d'une information supplémentaire qui est le coup de l'adversaire. On dit alors qu'il y a *information parfaite* [04].

2-2-2-2. Les différents types de jeux

Il existe différents types de jeux, on peut les classes dans deux sortes catégories, des jeux *coopératifs* et *non coopératifs:*

Au niveau des jeux *coopératifs*, on peut imaginer la coopération grâce à des contrats qui ne peuvent pas être remis en cause. Les joueurs peuvent également éventuellement transférer les gains d'un joueur à l'autre. Dans les cas extrêmes de coopération où les gains sont répartis équitablement entre les joueurs (on recherche alors une somme de gains maximum), on peut considérer les joueurs comme un joueur unique qui chercherait à dégager un *intérêt général* et qui ensuite répartirait les gains entre les joueurs.

Partant de l'hypothèse que chaque joueur garde sa liberté d'engagement, l'objectif de la théorie des jeux *non coopératifs* est de caractériser les issues possibles d'une interaction stratégique lorsque les joueurs abordent cette interaction de manière rationnelle, c'est-à-dire finalement de la manière la plus *égoïste* qui soit (ils veulent seulement maximiser leur propre bien).
Dans ce modèle, il est *impossible de communiquer* et de se concerter entre concurrents.
Parfois même si on supposait la concertation, on pourrait supposait le jeu comme non coopératif: en effet, il arrive que la perspective de rompre le contrat soit tellement forte (si elle génère des gains plus importants) que s'il n'y a pas une autorité suffisamment forte qui sanctionne fortement la rupture de contrat ou l'engagement (que certains joueurs ont passé avec d'autres joueurs avant le jeu), certains joueurs se comporteraient de manière égoïste et agiraient comme si le jeu n'était pas coopératif.

Dans les jeux non coopératifs, on peut encore distinguer 2 sous classes de jeu :

- *les jeux de lutte à l'état pur*, comme les jeux à somme nulle, qui peuvent être qualifiés de duel. On y regroupe tous les jeux dont la somme des gains est constante (Gain X + Gain Y+...=Constante) . Notons que par simple décalage des gains de X, Y.., on peut prendre la constante égale à 0.
Le gain de quelqu'un implique la perte pour quelqu'un d'autre : c'est le cas typique

Des jeux de société et des jeux de stratégies tels que le jeu de dames, le jeu d'échecs. Il n'y a donc pas de coopération possible. Ces jeux ont été étudiés préférentiellement au début de la théorie des jeux, car ils sont faciles à modéliser : ainsi, lorsqu'il y a 2 joueurs Gain X = - Gain Y.

- *les jeux de lutte et de coopération* où les intérêts entre les différents joueurs peuvent être divergents mais également convergents. La difficulté est que la perte de l'un n'est pas forcément le gain de l'autre : dans un jeu économique par exemple, il faut simultanément faire croître le gâteau et se le partager. Hélas, il arrive que ces objectifs soient difficiles à concilier et une compétition ruineuse peut détruire plus de richesses qu'elle n'en crée.
Plus brièvement, on peut retenir que les jeux coopératifs procurent en général des gains supérieurs aux jeux non-coopératifs mais ils sont beaucoup plus difficiles à modéliser,

Et c'est pour cela que la plupart des études portent sur les jeux non-coopératifs, bien qu'on leur reproche souvent d'être éloignés de la réalité (par exemple ils ne peuvent pas prendre en compte la possibilité d'alliance avec d'autres joueurs) [20].

2-2-3. Algorithmes de jeu

Les jeux en général sont représentés sous forme d'un espace d'états, a chaque état correspond une valeur calculée par une fonction appelée **fonction d'évaluation**. Résoudre Les jeux revient alors à proposer **une stratégie** permettant de parcourir l'espace d'états de manière à trouver l'état meilleur par rapport a notre configuration.

Garce a l'exemple qui suit on va vous présenter les principaux algorithmes utilise dans les jeux:

Exemple:

Soient JE (l'ordinateur) et LUI (l'adversaire).

- un sommet terminal correspond à la configuration gagnante
- tout autre sommet extrême correspond à une configuration perdante
- un sommet OÙ correspond à une configuration à partir de laquelle JE va jouer
- un sommet ET correspond à une configuration à partir de laquelle LUI va jouer

2-2-3-1. Algorithme minimax

JE cherche à Maximiser la fonction d'évaluation (MAX). LUI est supposé chercher à la Minimiser (MIN) (coups défavorables à JE) [04]

$f(P)$: Valeur de la fonction d'évaluation si P est un noeud extrême

$F(P)$: Valeur de retour associée à P si P est n'est pas extrémité

Si P1,P2, ... sont les successeurs directs de P (noeuds fils):

$F(P) = f(P)$ • si P est un noeud extrémité

$F(P) = \max_i F(P_i)$ • si P est un noeud MAX

$F(P) = \min_i F(P_i)$ • si P est un noeud MIN

2-2-3-2. Algorithme NEGMAX [Knuth & Moore 1975]

Le principe de cet algorithme consiste à éviter de traiter différemment les noeuds MIN et MAX. [04]

$F(n) = f(n)$ • si n extrémité

$F(n) = \max(-F(n_1), -F(n_2), \ldots)$ • sinon

2-2-3-3. Algorithme Alpha-Béta

Le gain de cet algorithme consiste à éviter de faire une recherche exhaustive jusqu'à la profondeur limite, en choisie alors les chemins qui comporte plus de valeur par rapport a notre fonction d'évaluation. [04]

- alpha = $-\infty$;
- bêta = $+\infty$;

```
int valeur(p, alpha, beta)
position p;
int alpha, beta;
{
  int i, t, nbre;

  /* trouver les coups possibles à partir de P: P1, P2... */
  trouver_successeurs(...);
  nbre=nbre_successeurs(...);

  if(nbre==0) return f(p);
  for(i=0;i<nbre;i++)
  {
    t= - valeurs(Pi, -beta, -alpha);
```

```
    if(t>alpha) alpha=t;
    if(alpha>=beta) return alpha;
  }

  return alpha;
}
```

- Améliorations possibles des algorithmes de jeu

Voici quelques points négliges par rapport les algorithmes précédent:

- Mettre dans le ``bon" ordre les successeurs (heuristique)
- Bibliothèque pour le début de partie (base de données qui contient tout les coups possibles)
- L'évaluation peut aussi tenir compte du nombre de bons coups possibles à partir de la position courante ⇒Evaluation confirmée [06]

2-3. Les Systèmes experts a Base de Règles:

2-3-1. Introduction

2-3-1-1. Qu'est-ce qu'un système expert

Un **système expert** est un logiciel qui reproduit le comportement d'un expert humain accomplissant une tâche intellectuelle dans un domaine précis. On peut souligner les points suivants :

- les systèmes experts sont généralement conçus pour résoudre des problèmes de *classification* ou de *décision* (diagnostic médical, prescription thérapeutique, régulation d'échanges boursiers, ...)
- les systèmes experts sont des outils de l'*intelligence artificielle*, c'est-à-dire qu'on ne les utilise que lorsqu'aucune méthode algorithmique exacte n'est disponible ou praticable
- un système expert n'est concevable que pour les domaines dans lesquels il existe des *experts humains*. Un expert est quelqu'un qui connaît un domaine et qui est plus ou moins capable de transmettre ce qu'il sait : ce n'est par exemple pas le cas d'un enfant par rapport à sa langue maternelle. [20]

Un système expert est composé de deux parties indépendantes :

- une **base de connaissances** elle-même composée d'une **base de règles** qui modélise la connaissance du domaine considéré et d'une **base de faits** qui contient les informations concernant le cas que l'on est en train de traiter
- un **moteur d'inférences** capable de raisonner à partir des informations contenues dans la base de connaissance, de faire des déductions, etc.

La figure suivante illustre les composants d'un SE :

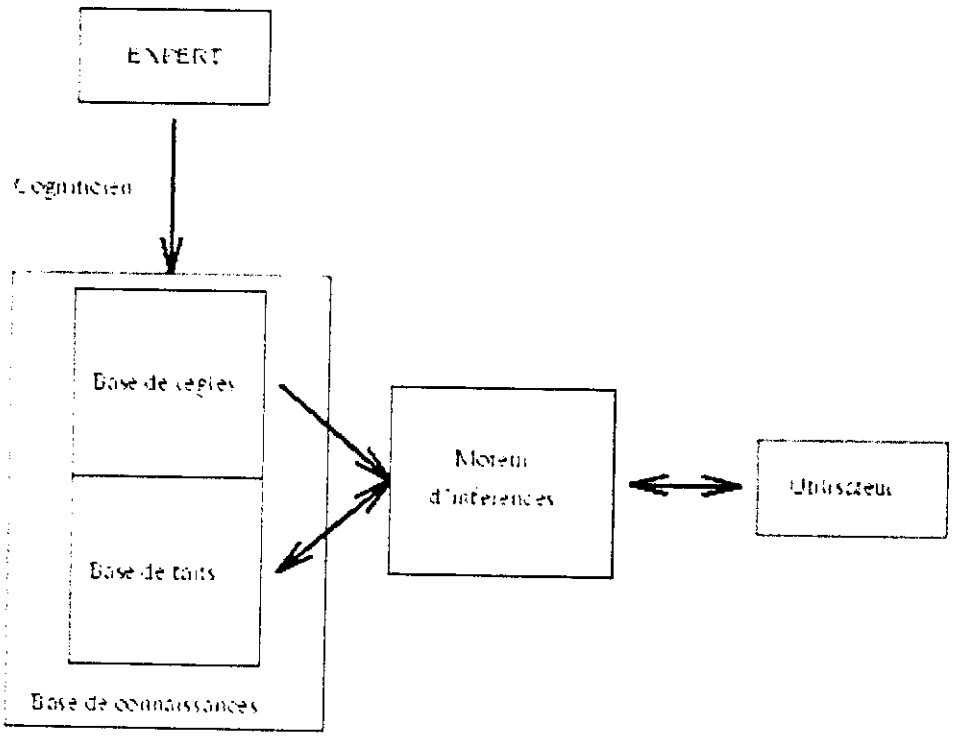


Figure 2-1 : Structure d'un système expert

Le rôle du **cogniticien** est de *soutirer* leurs connaissances aux experts du domaine et de *traduire* ces connaissances dans un formalisme se prêtant à un traitement automatique, c'est-à-dire en *règles*. Ces deux tâches sont aussi délicates l'une que l'autre. En effet, un expert est la plupart du temps inconscient de la majeure partie de son savoir ; et s'il arrive à en exprimer une partie, c'est souvent sous une forme qui ne se laisse pas facilement formaliser. De plus l'extraction de connaissances peut avoir un effet perturbant sur l'expert : il est bien connu que si l'on demande à un maître aux échecs de réfléchir à sa manière de jouer, on observera une baisse de ses performances dans les jours ou les semaines qui suivent cet effort d'introspection.

L'indépendance entre la base de connaissances et le moteur d'inférences est un élément essentiel des systèmes experts. Elle permet une représentation des connaissances sous forme purement *déclarative*, c'est-à-dire sans lien avec la manière dont ces connaissances sont utilisées. L'avantage de ce type d'architecture est qu'il est possible de faire évoluer les connaissances du système sans avoir à agir sur le mécanisme de raisonnement. Il en est de même pour nous : un accroissement ou une modification de nos connaissances n'entraîne pas nécessairement une restructuration en profondeur de nos mécanismes de fonctionnement.

Dans la réalité, les choses se passent de manière un peu moins idéale et il est souvent nécessaire d'*organiser* la base de connaissances, de réfléchir sur les stratégies d'utilisation des règles, etc.

Le système expert est souvent complété par des interfaces plus ou moins riches permettant un dialogue avec les utilisateurs, l'idéal étant une interface en langage naturel. [20]

2-3-1-2. Quelques systèmes experts classiques

Depuis leurs apparition les system experts n'ont pas cesser de s'évoluer dans plusieurs domaine, voila les plus fameux entre eux:

- DENDRAL, 1969, chimie, recherche la formule développée d'un corps organique à partir de la formule brute et du spectrogramme de masse du corps considéré.
- CRYSALIS, 1979, chimie, recherche la structure de protéines à partir de résultats d'analyse cristallographique.
- MOLGEN, 1977, biologie, engendre un plan de manipulations génétiques en vue de construire une entité biologique donnée.
- PROSPECTOR, 1978, géologie, aide le géologue à évaluer l'intérêt d'un site en vue d'une prospection minière. (1600 règles)
- AM, 1977, mathématiques, proposition de conjecture, de concepts intéressants. (500 règles)
- MUSCADET, 1984, mathématiques, démonstration de théorèmes (EVT).
- MYCIN, 1974, médecine, système d'aide au diagnostique et au traitement de maladies bactériennes du sang

[20]

2-3-1-3. Systèmes experts et générateurs de systèmes experts

Un système expert concerne un domaine précis d'application. Un *système expert général* (ou *générateur de systèmes experts*) est un outil de développement de systèmes experts particuliers. Un système expert général contient un moteur d'inférences et un système de représentation des connaissances (par exemple, s'il s'agit d'un générateur d'ordre 0, la possibilité de définir et de manipuler des faits booléens). [20]

Quelques exemples de systèmes experts généraux :

- VP-Expert (pour des applications peu complexes)
- SNARK (beaucoup plus complet, mais date un peu)
- GURU (idem)
- PROLOG (c'est plus un langage de programmation qu'un générateur)
- SMECI (un générateur contemporain)

2-3-2. La base de connaissances :

2-3-2-1. La base de faits

Mémoire de travail

La *base de faits* est la **mémoire de travail** du système expert. Elle est variable au cours de l'exécution et vidée lorsque l'exécution se termine. Au début de la session, elle contient ce que l'on sait du cas examiné avant toute intervention du moteur

d'inférences. Puis elle est complétée par les faits déduits par le moteur ou demandés à l'utilisateur.

Par exemple, dans le domaine médical, la base de faits pourra contenir une liste de symptômes en début de session et un diagnostic lorsque celle-ci se terminera. [15]

Le type d'un fait

Les faits peuvent prendre des formes plus ou moins complexes. Nous n'envisagerons que des faits élémentaires dont les valeurs possibles sont

- **booléennes** : vrai, faux
- **symboliques** : c'est-à-dire appartenant à un domaine fini de symboles
- **réelles** : pour représenter les faits continus.

Par exemple, *actif* est un fait booléen, *profession* est un fait symbolique et *rémunération* est un fait réel.

Un système expert qui n'utilise que des faits booléens est dit d'*ordre 0*.

Un système expert qui utilise des faits symboliques ou réels, sans utiliser de variables, est d'*ordre 0+*.

Un système utilisant toute la puissance de la logique du premier ordre est d'*ordre 1*.

Les formules ou conditions

Dans un système expert d'ordre 0, on pourra par exemple écrire des formules de la forme :
actif ou ¬ *actif*

Dans un système d'ordre 0+, on pourra trouver les formules :

actif **et** (*profession* ≠ *medecin* **ou** *remuneration* ≤ 20000)

Dans un système d'ordre 1, on pourra trouver :

$\exists X$ *maladie*(X) **et** $X \neq$ grippe **et** *symptome*(X) = *forteFievre*

Ces formules sont appelées *conditions* lorsqu'elles servent à déclencher des règles. On remarque que les faits booléens peuvent être interprétés comme des formules puisqu'ils possèdent une valeur de vérité. [15]

Métafaits et Métavaleurs

Pour qu'un système expert puisse modéliser un raisonnement humain, il est indispensable qu'il puisse raisonner sur ses propres raisonnements, réfléchir aux faits qu'il manipule, aux formules qu'il peut construire, etc. Autrement dit, il n'est pas suffisant que le système ait des connaissances, il faut aussi qu'il ait des **métaconnaissances**.

Il faut par exemple qu'un système expert puisse savoir si une valeur a été attribuée à un fait. Dans la négative, cette valeur pourra être demandée à l'utilisateur. Mais si l'utilisateur ne peut pas répondre, il faudra que le système puisse le savoir afin de ne pas poser éternellement la même question. La seule manière d'attribuer une valeur à un tel fait sera alors de la déduire d'autres faits. [15]

On dira que la valeur d'un fait est :

- *connue* si une valeur lui a été attribuée
- *inconnue* si aucune valeur ne lui a été attribuée et si aucune question à son sujet n'a été posée à l'utilisateur
- *indéterminée* si le système ne lui a attribué aucune valeur et et si l'utilisateur a répondu << je ne sais pas >> à une question concernant sa valeur.

Ainsi,
Valeur (*profession*)
Est un *métafait* symbolique et
Valeur (*profession*)= *connue*
Est une *métacondition.*

De manière analogue, tous les faits ne doivent pas faire l'objet d'une question à la personne concernée. Il n'est pas envisageable qu'un médecin demande à son patient
<< Quelle maladie avez-vous ? >>
ni qu'un juge demande à la personne comparaissant devant lui
<< À quelle peine dois-je vous condamner ? >>

Un système expert doit donc savoir si un fait est *demandable* ou non. Donc,
Demandable (*diagnostic*)
Est un métafait booléen (et une métacondition).

2-3-2-2. La base de règles

Elle rassemble la connaissance et le savoir-faire de l'expert. Elle n'évolue donc pas au cours d'une session de travail.

Une **règle** est de la forme
Si <conjonction de conditions> **alors** <conclusion>

où les conclusions sont de la forme
<Fait> = <valeur>.

Exemple :

si population > 200000 **et** ville-universitaire **alors** cinéma ArtEtEssai=*vrai*

si revenu-imposable = *connu* **et** quotient-familial = *connu* **alors** calculerMontantImpot=*vrai*

Le dernier exemple montre comment un système expert peut être utilisé en association avec des programmes classiques. On peut supposer que le passage à la valeur *vrai* du fait booléen *calculer Montant Impôt* déclenche une procédure calculant le montant de l'impôt en question et l'attribuant au fait réel *Montant Impôt*.

Une base de règles est un ensemble de règles et sa signification logique est la *conjonction* de la signification logique de chacune des règles. En particulier, on peut aisément coder dans le formalisme précédent des règles de la forme [20]
si A **ou** B **alors** C
ou
si A **alors** B **et** C
Il n'en est par contre pas de même de
si A **alors** B **ou** C

2-3-2-3. Les métarègles et la métaconnaissance

De même que dans un système expert, la connaissance est traduite en règles, la métaconnaissance s'exprime par des métarègles (c'est-à-dire des règles sur la manière d'utiliser les règles).

On trouve par exemple dans MYCIN les métarègles suivantes:

1. **si** on recherche une thérapie **alors** considérer dans cet ordre les règles qui permettent de :
   - acquérir les informations cliniques sur le patient
   - trouver quels organismes, s'il en existe, sont causes de l'infection
   - identifier les organismes les plus vraisemblables
   - trouver tous les médicaments potentiellement utiles
   - choisir les plus adaptés en plus petit nombre

2. **si** le patient est un hôte à risque **et** s'il existe des règles mentionnant des pseudonymies dans une prémisse **et** s'il existe des règles mentionnant des klebsiellas dans une prémisse **alors** il est probable qu'il faille utiliser les premières avant les secondes

3. **si** le site de la culture est non stérile **et** il existe des règles mentionnant dans leurs prémisses un organisme déjà rencontré auparavant chez le patient et qui peut être le même que celui dont on recherche l'identité **alors** il est sûr qu'aucune d'entre elles ne peut servir

L'organisation d'une base de connaissances au moyen de méta-règles reste essentiellement déclarative, contrairement à toute organisation basée sur une structuration *a priori* de l'ensemble des règles (écrire les règles dans un ordre donné, ...). [20]

2-3-2-4. La représentation des connaissances incertaines

Un des plus grands problèmes que rencontre le cogniticien lorsqu'il tente de formaliser le savoir d'un expert, c'est que celui-ci est capable de raisonner sur des *connaissances incertaines* et qu'on ne dispose que de très peu d'outils pour rendre compte de cette capacité.

Un médecin par exemple n'est jamais totalement sûr que tel symptôme est signe de telle maladie, que tel médicament sera supporté par le malade, que le malade guérira, etc.

On peut reconnaître globalement un objet sans être capable d'identifier à 100 pour cent chacun de ses détails.

L'observation du temps qu'il fait ne permet pas de savoir avec certitude s'il pleuvra ou non une heure plus tard.

On peut utiliser la *théorie des probabilités* pour définir le **degré de vraisemblance** d'un fait. De nombreux générateurs de systèmes experts offrent la possibilité aux utilisateurs de nuancer leur certitude concernant un fait en leur associant un degré de vraisemblance. [20]

<< Reconnaissez-vous votre agresseur sur cette photo ? >>
<< Oui, enfin c'est-à-dire, non. En fait, je le reconnais à 72 % >>
Cette technique permet aussi de graduer une sensation.

<< Avez-vous très mal à la tête ? Ou seulement, un peu mal >>
<< Sur une échelle qui associerait 0 au fait de ne pas avoir mal à la tête et 1 au fait de ne pas pouvoir le supporter, je dirai que j'ai mal à la tête avec un coefficient 0,45 >>

Les exemples qui précèdent révèlent une des principales difficultés de cette méthode : il n'est pas raisonnable d'attendre d'un être humain, expert ou non, qu'il puisse définir avec précision de tels degrés de vraisemblance.

Un autre problème concerne la combinaison des coefficients de vraisemblance. Si les faits A et B sont connus respectivement avec les coefficients de vraisemblance p et q et si la règle
Si A **et** B **alors** C
Figure dans la base de règles, que peut-on déduire pour le fait C ? Qu'il est vrai avec le degré de vraisemblance pq ? Mais ceci exige que les faits A et B soient *indépendants*. Comment s'en assurer ? Et si ils ne le sont pas ?

Et comment rendre compte par une règle d'une connaissance du type

Le fait que A soit vrai rend B plus vraisemblable ?

De nombreux travaux en Intelligence Artificielle sont consacrés à ces questions. On les regroupe généralement sous le vocable ``recherche sur la logique naturelle" puisqu'il s'agit de comprendre en quoi nos raisonnements courants peuvent être décrits et formalisés par des règles logiques. On peut citer

- la **logique floue** qui explore la voie esquissée plus haut consistant à associer des coefficients de vraisemblances à des faits
- les **logiques modales** qui introduisent des opérateurs modaux comme *``il est possible que''* ou *``il est nécessaire que''*
- les **logiques non monotones** qui étudient la possibilité de réviser un raisonnement (si on me dit que TITI est un oiseau, j'en déduis qu'il vole mais si l'on ajoute que TITI est une autruche, je révise mon inférence). [20]

2-3-3. Les moteurs d'inférences a base de règles

Un moteur d'inférences est un mécanisme qui permet d'inférer des connaissances nouvelles à partir de la base de connaissances du système (Les règles du system).

On distingue essentiellement trois modes principaux de fonctionnement des moteurs d'inférences on traitera que le premier par ce que c'est lui qui est utiliser dans notre application:

- le *chaînage avant*
- le *chaînage arrière*
- et le *chaînage mixte*

On remarquera que les moteurs d'inférence décrits ci-dessous le sont indépendamment de tout domaine d'application. Cette séparation entre connaissance et raisonnement est essentielle pour les systèmes experts. [15]

2-3-3-1. Le chaînage avant

Le mécanisme du chaînage avant est très simple : pour déduire un fait particulier, on déclenche les règles dont les prémisses sont connues jusqu'à ce que le fait à déduire soit également connu ou qu'aucune règle ne soit plus déclenchable. [15]

Plus précisément, soit *BF* une base de faits, *BR* une base de règles (ne comportant que des faits *booléens positifs*) et *Fait* le fait que l'on cherche à établir, l'algorithme suivant calcule si *Fait* peut être déduit ou non de la base de connaissances. [20]

ALGORITHME DU CHAINAGE AVANT
ENTREE : BF, BR, F
DEBUT
  TANT QUE F n'est pas dans BF ET
  QU'il existe dans BR une règle applicable FAIRE
    choisir une règle applicable R (étape de résolution de conflits, utilisation d'heuristiques, de métarègles)
    BR = BR - R (désactivation de R)
    BF = BF union concl(R) (déclenchement de la règle R, sa conclusion est rajoutée à la base de faits)
  FIN DU TANT QUE
  SI F appartient à BF ALORS
    F est établi
  SINON
    F n'est pas établi

On remarque que l'algorithme précédent n'indique pas comment choisir une règle applicable. C'est à ce niveau que la métaconnaissance du domaine intervient et permet de définir une stratégie de choix.

Exemple :

Soit $BF = \{B,C\}$, $Fait = H$ et BR composée des règles :

- **Si** B **et** D **et** E **alors** F
- **Si** G **et** D **alors** A
- **Si** C **et** F **alors** A
- **Si** B **alors** X
- **Si** D **alors** E
- **Si** X **et** A **alors** H
- **Si** C **alors** D
- **Si** X **et** C **alors** A
- **Si** X **et** B **alors** D

L'algorithme précédent appliqué à ces paramètres prouve que H se déduit de la base de connaissances.

2-3-4. L'utilisation des SE dans les jeux

Pour répondre aux besoins de décisions stratégiques, les concepteurs de jeux utilisent généralement des systèmes experts.
Des règles permettent de définir un ensemble de stratégies servant de base à la prise de décision en fonction des situations du jeu.
Meme face à des situations non prévues ou ambiguës, le système peut être amené à prendre des décisions non programmer. Il permet de prendre des décisions plus adaptées face à des situations imprévues et surtout d'avoir des comportements plus humains.
Voici quelques exemples:
Dans S.W.A.T., de Yosemite Software, un terroriste plutôt nerveux peut décider de fuir ou d'attaquer.

Figure 2-2:S.W.A.T

défi

Figure 2-3:Age of Empires

Les systèmes experts sont une des applications de l'intelligence artificielle qui ont quitté les laboratoires de recherche pour être utilisées dans le *monde de l'entreprise*. De nombreux systèmes experts ont été implantés avec succès pour résoudre des problèmes concrets.

Mais les grandes difficultés que l'on rencontre lorsqu'on cherche à *extraire* des experts leur connaissance, puis lorsqu'on tente de *formaliser* ces connaissances dégagées à grand peine, sont peut être signes d'une faiblesse intrinsèque des systèmes experts. La *possibilité* de concevoir des systèmes experts repose en effet sur une hypothèse psychologique dont il n'a pas été question ci-dessus : la structuration de nos aptitudes en règles symboliques parfaitement identifiables. Majoritaire chez les psychologues et chercheurs en intelligence artificielle dans les années 70, cette hypothèse est de plus en plus battue en brèche et l'on envisage maintenant des modèles de savoir qui n'y ont plus recours (modèles connexionistes, systèmes adaptatifs, etc.). Il n'est par ailleurs pas exclu que les programmes à venir utilisent Les systèmes experts dans des autres domaines (les jeux) avec des techniques plus adéquates pour tirer le plus partis.

2-4. Conclusion

Apres avoir décrit l'intelligence artificielle dans le domaine des jeux, maintenant, on présentera l'outil de développement de notre jeu- Java 2 Micro Edition-.

Chapitre 3 :

La Java 2 Micro Edition

3- La Java 2 Micro Edition:

J2ME est la plate-forme java pour développer des applications sur des appareils mobiles tel que des PDA, des téléphones cellulaires, des terminaux de points de vente, des systèmes de navigations pour voiture, ...

C'est une sorte de retour aux sources puisque Java avait été initialement développé pour piloter des appareils électroniques. [01]

Ce chapitre contient plusieurs sections :

- Présentation de J2ME : présentation rapide de J2ME
- Les configurations : présentation des deux configurations sur lesquels la plate-forme J2ME repose
- Les profiles : présentation des profils qui enrichissent les configurations pour un type machines ou à une fonctionnalité spécifique

3-1. Présentation de J2ME

Historiquement, Sun a proposé plusieurs plate-formes 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.

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

En 1999, Sun propose 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, J2ME utilise des machines virtuelles différentes. Certaines classes de base de l'API sont communes avec cependant de nombreuses omissions dans l'API J2ME.

L'ensemble des appareils sur lequel peut s'exécuter une application écrite avec J2ME est tellement vaste et disparate que J2ME est composé de plusieurs parties : les configurations et les profiles qui sont spécifiés par le JCP. J2ME propose donc une architecture modulaire.

Chaque configuration peut être utilisée avec un ensemble de packages optionnels qui permet d'utiliser des technologies particulières (Bluetooth, services web, lecteur de codes barre, etc ...). Ces packages sont le plus souvent dépendant du matériel.

L'inconvénient de ce principe est qu'il déroge à la devise de Java "Write Once, Run Anywhere". Ceci reste cependant partiellement vrai pour des applications développées pour un profile praticulier. Il ne faut cependant pas oublier que les types de machines cibles de J2ME sont tellement différents (du téléphone mobile au set top box), qu'il est sûrement impossible de trouver un dénominateur commun. Ceci associé à l'explosion du marché des machines mobiles explique les nombreuses évolutions en cours de la plate-forme. [01]

J2ME est la plate-forme Java la plus récente.

De plus d'informations peuvent être obtenues sur les deux sites de Sun :

- http://wireless.java.sun.com/
- http://java.sun.com/j2me/

Et sur les sites:

- http://www.javamobiles.com/
- http://www.microjava.com/

3-2. Les Configurations

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)
- CDC (Connected Device Configuration).

La CLDC 1.0 est spécifiée dans la JSR 030 : elle concerne des appareils possédant des ressources faibles (moins de 512 Kb de RAM, faible vitesse du processeur, connexion réseau limitée et intermittente) et une interface utilisateur réduite (par exemple un téléphone mobile ou un PDA "bas de gamme"). Elle s'utilise sur une machine virtuelle KVM. La version 1.1 est le résultat des spécifications de la JSR 139 : une des améliorations les plus importantes est le support des nombres flottants.

La CDC est spécifié dans la JSR 036 : elle concerne des appareils possédant des ressources plus importantes (au moins 2Mb de RAM, un processeur 32 bits, une meilleure connexion au réseau), par exemple un settop box ou certains PDA "haut de gamme". Elle s'utilise sur une machine virtuelle CVM [01]

3-3. Les Profiles

Lorsqu'une configuration définit le fondement d'une application, un profil en fournit la structure. Les Profils définissent l'ensemble des API à utiliser dans une application J2ME et sont conçus spécialement pour chaque configuration.

Sun propose deux profils de référence J2ME : le profil Foundation et le profil Mobile Information
Device Profile (MIDP).

· Le profil Foundation est destiné à la configuration CDC. Les développeurs qui utilisent ce profil ont accès à une implémentation complète des fonctionnalités de J2SE.

· Le profil MIDP est destiné à la configuration CLDC. Il prend en charge un nombre limité des classes de J2SE
et définit des classes d'entrée/sortie et d'interface spécialisées pour une Configuration CLDC. [01]

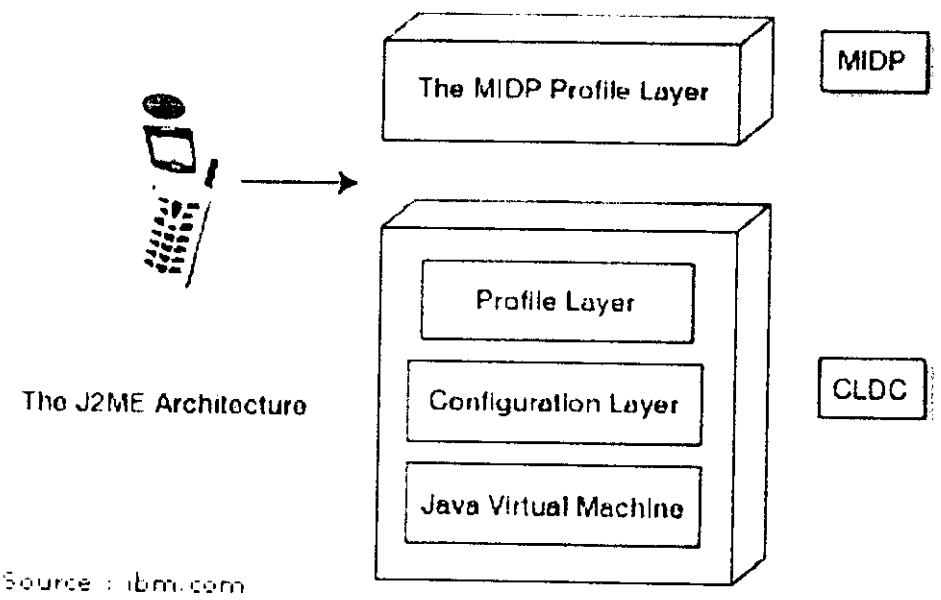


Figure 3-1:Architecture J2ME

Il existe plusieurs profiles:

		Application
		Personal Profile
Application Midlet	Application pour Palm	Personal Basis Profile
MIDP	MIDP for Palm OS	Foundation Profile
CLDC	CLDC	CDC
KVM	KVM	CVM

Figure 3-2:profiles J2ME

MIDP est un profile standard qui n'est pas défini pour une machine particulière mais pour un ensemble de machines embarquées possédant des ressources et une interface graphique limitée.

Sun a développé un profil particulier nommé KJava pour le développement spécifique sur Palm. Ce profile a été remplacé par un nouveau profil nommé MIDP for Palm OS.

Le Foundation Profile est un profil de base qui s'utilise avec CDC. Ce profil ne permet pas de développer des IHM. Il faut lui associer un des deux profils suivants :

- le Personal Basic Profile permet le développement d'application connectée avec le réseau
- le Personal Profile est un profil qui permet le développement complet d'une IHM et d'applet grâce à AWT.

PersonalJava est remplacé par le Personal Profile.

Le choix du ou des profils utilisés pour les développements est important car il conditionne l'exécution de l'application sur un type de machine supporté par le profil.

Cette multitude de profils peut engendrer un certain nombre de problème lors de l'exécution d'une application sur différents périphériques car il n'y a pas la certitude d'avoir à disposition les profils nécessaires. Pour résoudre ce problème, une spécification particulière issue des travaux de la JSR 185 et nommée Java Technology for the Wireless Industry (JTWI) a été développée. Cette spécification impose aux périphériques qui la respectent de mettre en oeuvre au minimum : CLDC 1.0, MIDP 2.0, Wireless Messaging API 1.1 et Mobile Media API 1.1. Son but est donc d'assumer une meilleure compatibilité entre les applications et les différents téléphones mobiles sur lesquelles elles s'exécutent. [02]

3.4. Les Midlets

Les applications créées avec MIDP sont des **midlets** : ce sont des classes qui héritent de la classe abstraite **javax.microedition.midlet.Midlet.** Cette classe permet le dialogue entre le système et l'application.
Elle possède trois méthodes qui permettent de gérer le cycle de vie de l'application en fonction des trois états possibles (active, suspendue ou détruite) :

startApp() : cette méthode est appelée à chaque démarrage ou redémarrage de l'application
pauseApp() : cette méthode est appelée lors de la mise en pause de l'application
destroyApp() : cette méthode est appelée lors de la destruction de l'application
Ces trois méthodes doivent obligatoirement être redéfinies. [07]

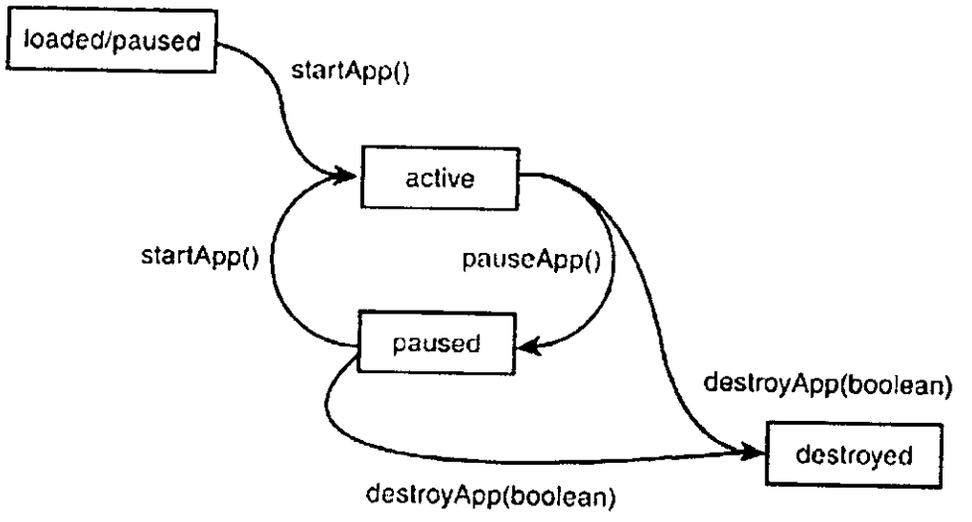


Figure 3-3:Cycle de vie d'une MIDlet

Exemple d'une Midlet

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Test extends MIDlet {

  public Test() {
  }

  public void startApp() {
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }
}
```

Le cycle de vie d'une MIDlet est semblable à celui d'une applet.

3-5. L'interface utilisateur

Les possibilités concernant l'IHM de MIDP sont très réduites pour permettre une exécution sur un maximum de machines allant du téléphone portable au PDA. Ces machines sont naturellement et physiquement pourvues de contraintes forte concernant l'interface qu'ils proposent à leurs utilisateurs.
Avec le J2SE, deux API permettent le développement d'IHM : AWT et Swing. Ces deux API proposent des composants pour développer des interfaces graphiques riches de fonctionnalités avec un modèle de gestion des événements complet. Ils prennent en compte un système de

pointage par souris, avec un écran couleur possèdent de nombreuses couleurs et une résolution importante.
Avec MIDP, le nombre de composants et le modèle de gestion des événements sont spartiates. Il ne prend en compte qu'un écran tactile souvent monochrome ayant une résolution très faible. Avec un clavier limité en nombres de touches et dépourvu de système de pointage, la saisie de données sur de tels appareils est particulièrement limitée.
L'API pour les interfaces utilisateurs du MIDP est regroupée dans le package **javax.microedition.lcdui**.
Elle se compose des Classes d'Affichages, des éléments de hauts niveaux et des éléments de bas niveaux.

3-5-1. Les Classes d'Affichages

3-5-1-1. La classe Display

Pour pouvoir utiliser les éléments graphiques, il faut obligatoirement obtenir un objet qui encapsule l'écran. Un tel objet est du type de la classe Display. Cette classe possède des méthodes pour afficher les éléments graphiques.
La méthode statique getDisplay() renvoie une instance de la classe Display qui encapsule l'écran associé à la midlet fournie en paramètre de la méthode.[10]
Exemple:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Hello extends MIDlet {

 private Display display;

 public Hello() {
  display = Display.getDisplay(this);
 }

 public void startApp() {
 }

 public void pauseApp() {
 }

 public void destroyApp(boolean unconditional) {
 }
}
```

3-5-1-2.Les Autres Classes

Les éléments de l'interface graphique appartiennent à une hiérarchie d'objets : tous les éléments affichables héritent de la classe abstraite **Displayable**.

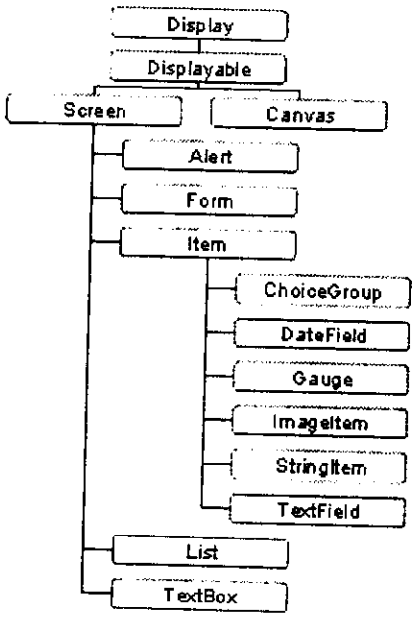


Figure 3-4:hiérarchie des Classes d'IU

La classe **Screen** est la classe mère des éléments graphiques de haut niveau. La classe Canvas est la classe mère des éléments graphiques de bas niveau.
Il n'est pas possible d'ajouter directement un élément graphique dans un Display sans qu'il soit inclus dans un objet héritant de Displayable.
Un seul objet de type Displayable peut être affiché à la fois. La classe Display possède la méthode getCurrent() pour connaître l'objet courant affiché et la méthode setCurrent() pour afficher l'objet fourni en paramètre. [10]

3-5-2. L'interface Utilisateur de Haut Niveau

3-5-2-1. La classe TextBox

Ce composant permet de saisir du texte. [11]
Exemple:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Hello extends MIDlet {

 private Display display;
 private TextBox textbox;

 public Hello() {
  display = Display.getDisplay(this);
  textbox = new TextBox("", "Bonjour", 20, 0);
 }

 public void startApp() {
  display.setCurrent(textbox);
 }

 public void pauseApp() {
```

```
  }

  public void destroyApp(boolean unconditional) {
  }
}
```

Résultat :

sur l'émulateur de téléphone mobile

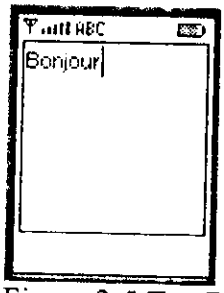


Figure 3-5:TextBox

3-5-2-2. La classe List

Ce composant permet la sélection d'un ou plusieurs éléments dans une liste d'éléments. [11]

Exemple:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Test extends MIDlet {
 private Display display;
 private List liste;

 protected static final String[] elements = {"Element 1",
                                             "Element 2",
                                             "Element 3",
                                             "Element 4"};

 public Test() {
  display = Display.getDisplay(this);
  liste = new List("Selection", List.EXCLUSIVE, elements, null);;
 }

 public void startApp() {
  display.setCurrent(liste);
 }

 public void pauseApp() {
 }

 public void destroyApp(boolean unconditional) {
 }
}
```

Résultat :

sur l'émulateur de téléphone mobile

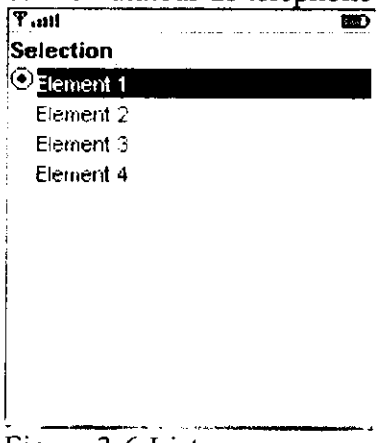


Figure 3-6:List

3-5-2-3. La classe Form

La classe **From** permet d'insérer dans l'élément graphique qu'elle représente d'autres éléments graphiques : cette classe sert de conteneurs. Les éléments insérés sont des objets qui héritent de la classe abstraite Item. [12]

Exemple:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Hello extends MIDlet {

 private Display display;
 private Form mainScreen;

 public Hello() {
  display = Display.getDisplay(this);
 }

 public void startApp() {
  mainScreen = new Form("Hello");
  mainScreen.append("Bonjour");
 display.setCurrent(mainScreen);
 }

 public void pauseApp() {
 }

 public void destroyApp(boolean unconditional) {
 }
}
```

3-5-2-4. La classe Item

La classe **javax.microedition.lcdui.Item** est la classe mère de tous les composants graphiques qui peuvent être insérés dans un objet de type Form.

Cette classe définit seulement deux méthodes, getLabel() et setLabel() qui sont le getter et le setter pour la propriété label. [13]
Il existe plusieurs composants qui héritent de la classe Item

Classe	Rôle
ChoiceGroup	sélection d'un ou plusieurs éléments
DateField	affichage et saisie d'une date
Gauge	affichage d'une barre de progression
ImageItem	affichage d'une image
StringItem	affichage d'un texte
TextField	saisie d'un texte

Figure 3-7:composants de la classe Item

Exemple:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Hello extends MIDlet {

 private Display display;
 private Form form;
 private ChoiceGroup choiceGroup;
 private DateField dateField;
 private DateField timeField;
 private Gauge gauge;
 private StringItem stringItem;
 private TextField textField;

 public Hello() {
  display = Display.getDisplay(this);
  form = new Form("Ma form");

  String choix[] = {"Choix 1", "Choix 2"};
  stringItem = new StringItem(null,"Mon texte");
  choiceGroup = new ChoiceGroup("Sélectionner",Choice.EXCLUSIVE,choix,null);
  dateField = new DateField("Heure",DateField.TIME);
  timeField = new DateField("Date",DateField.DATE);
  gauge = new Gauge("Avancement",true,10,1);
  textField = new TextField("Nom","Votre nom",20,0);

  form.append(stringItem);
  form.append(choiceGroup);
  form.append(timeField);
  form.append(dateField);
  form.append(gauge);
  form.append(textField);
}
```

```
  public void startApp() {
    display.setCurrent(form);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }
}
```

Résultat sur l'émulateur de téléphone mobile :

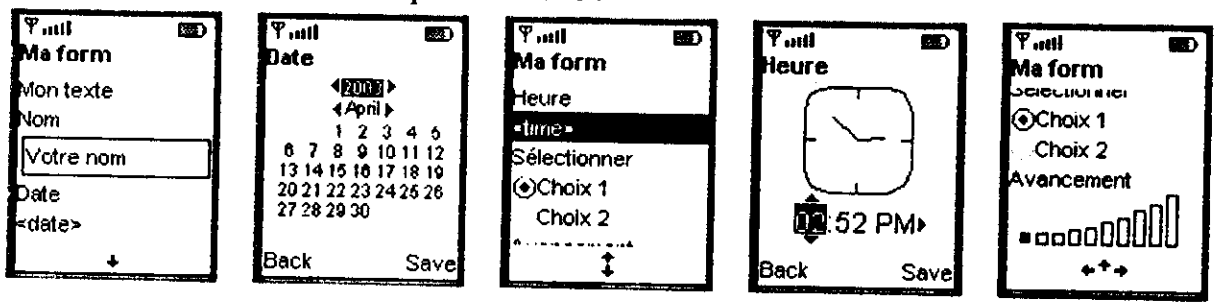


Figure 3-8:Les Composants de la classe Item

3-5-2-5. La classe Alert

Cette classe permet d'afficher une boite de dialogue pendant un temps déterminé.
Elle possède deux constructeurs :
Un demandant le titre de l'objet
Un demandant le titre, le texte, l'image et le type de l'image
Elle possède des getters et des setters sur chacun de ces éléments.
Pour préciser le type de la boîte de dialogue, il faut utiliser une des constantes définies dans la classe AlertType dans le constructeur ou dans la méthode setType() :

Constante	type de la boite de dialogue
ALARM	informer l'utilisateur d'un événement programmé
CONFIRMATION	demander la confirmation à l'utilisateur
ERROR	informer l'utilisateur d'une erreur
INFO	informer l'utilisateur
WARNING	informer l'utilisateur d'un avertissement

Figure 3-9:Les Types d'Alerts

Pour afficher un objet de type Alert, il faut utiliser une version surchargée de la méthode setCurrent() de l'instance de la classe Display. Cette version nécessite deux paramètres : l'objet Alert à afficher et l'objet de type Displayable qui sera affiché lorsque l'objet Alert sera fermé.

La méthode setTimeout() qui attend un entier en paramètre permet de préciser la durée d'affichage en milliseconde de la boîte de dialogue. Pour la rendre modale, il faut lui passer le parametre Alert.FOREVER.

Exemple:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Test extends MIDlet {

 private Display display;
private Alert alert;
 private Form form;

 public Test() {
  display = Display.getDisplay(this);
  form = new Form("Hello");
  form.append("Bonjour");

 alert = new Alert("Erreur", "Une erreur est survenue", null, AlertType.ERROR);
 alert.setTimeout(Alert.FOREVER);
}

public void startApp() {
 display.setCurrent(alert, form);
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
 }
}
```

Résultat sur l'émulateur de téléphone mobile:

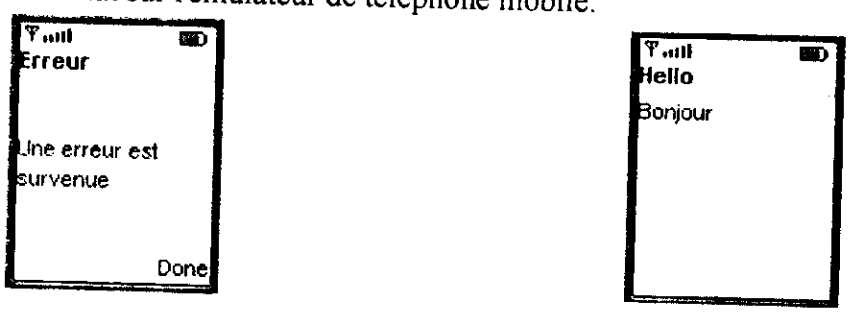


Figure 3-10:Alert

3-5-2-6. La Gestion des Evénements

Les interactions entre l'utilisateur et l'application se concrétisent par le traitement d'événements particuliers pour chaque action.[11]
MIDP définit interface de type Listener pour la gestion des événements :

Interface	Rôle
CommandListener	Listener pour une activation d'une commande
ItemStateListener	Listener pour un changement d'état d'un composant(modification du texte d'une zone de texte, ...)

Figure 3-11:Type Listener pour la gestion des événements

3-5-3. L'interface Utilisateur de Bas Niveau

L'API de l'interface utilisateur de bas niveau donne accès a l'écrans du terminal et aux événement associer aux touches et aux pointeur .Aucun composant d'interface utilisateur est disponible a ce niveau ,Notre application doit explicitement dessiner chaque composant même les commandes. [02]

3-5-3-1. La classe Canvas

La classe Canvas est la sous classe de Displayable qui implémente l'API de bas niveau. Le méthode abstraite paint() doit être implémenter .Les implémentations par défaut des méthodes de gestion des événements sont vides par défaut .L'Application n'a donc a surcharger que les méthodes utilisées. [02]

La Taille du Canvas peut être obtenue par les méthodes suivantes :

```
int getheight();
int getwidth();
```

3-5-3-2. La classe Graphics

La classe **Graphics** permet de reproduire des graphiques 2D.Elle est similaire a la class **java.awt.Graphics** de J2SE.La production de ses graphiques peut être faite directement a l'écrans ou en passant par un tampon d'image.
Puisque tout trace de graphique doit se faire via un objet Graphics .Vous devez au préalable en obtenir une référence . Il existe pour cela deux moyens. [02]

- Utiliser le paramètre de la méthode **paint()** .Les modifications sont alors visibles immédiatement.
- Utiliser **getGraphics()** qui permet de tracer des graphiques dans un tampon et de n'afficher le résultat qu'ensuite.

Le système des Coordonnées:

Comme illustré dans la figure suivante le système des coordonnées a pour origine le point supérieur gauche. [08]

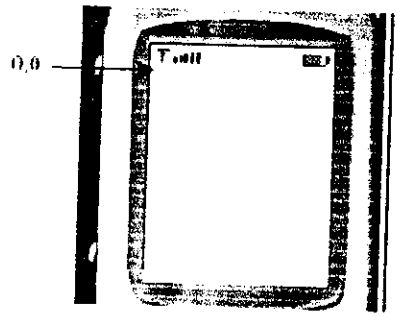


Figure 3-12:Point d'origine

Ce system a pour particularité que, dans la trace des traits, la largeur d'un trait est de un pixel par défaut. Le comportement est différent selon que vous tracez une forme pleine ou vide .Ainsi c'est le méthode qui trace un rectangle plein. [08]

Fillrect (1, 1, 3, 3);

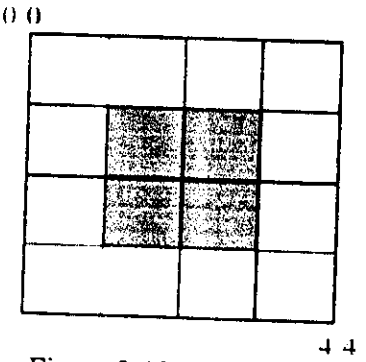


Figure 3-13:Rectangle 1

Remplit les pixels compris dans la zone (1 , 1) et (3 , 3) ,comme est dessine dans la figure précédente.

drawRect (1, 1, 4, 4);

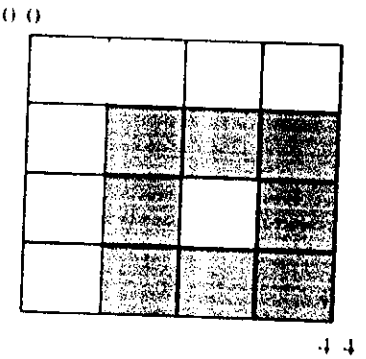


Figure 3-14:Rectangle 2

Active les pixels se trouvant dans les zones (1 , 1) et (4 , 4) ainsi que ceux se trouvant a la droite et en dessous du trace , soit les zones (1,1) et (4 , 4) comme illustre la figure précédente.

Trace de Ligne

Vous pouvez tracer une ligne :

void drawline(int x1, int y1, int x2, int y2);

Une ligne est race de position (x1, y1) a la position (x2, y2). La couleur et le style en cours sont utilises. [08]

Trace de Rectangle

Vous pouvez trace un rectangle vide

void drawRect(int x1, int y1, int x2, int y2);

Ou un rectangle plein avec:

void fillRect(int x1, int y1, int x2, int y2);

Vous pouvez également tracer un rectangle arrondi, respectivement vide et plein:

void drawRoundRect(int x, int y, int width, int height , int arcwidth, int archeight);
void fillRoundRect(int x, int y, int width, int height , int arcwidth, int archeight);

Trace d'un arc

Le trace d'un arc est effectué par ces méthodes: [09]

void drawArc(int x, int y, int width, int height , int startAngle, int arcAngle);
void fillArc(int x, int y, int width, int height , int startAngle, int arcAngle);

Les Couleurs

Les couleurs sont représentes sur 24 bits ,8 bits étant repartis pour chaque composante de couleur rouge, vert et bleu .Cependant .le terminal peut supporter moins de couleurs ou tout simplement des niveaux de gris.

Pour savoir si le terminal supporte les couleur utilise cette méthode:

boolean isColor();

le cas échant ,le nombre de couleurs supportes peut être obtenu par:

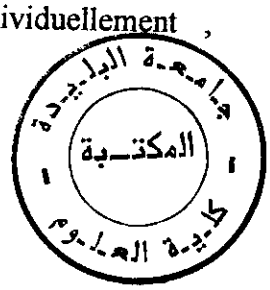
int numColors();

la valeur de la couleur est retourne par cette méthode:

int getColor();

les différentes composantes de couleur peuvent être retournées individuellement , respectivement pour le rouge , le vert et le bleu:

int getRedComponent();
int getGreenComponent();
int getBlueComponent();



Si la couleur n'est pas supportes vous pouvez obtenir le niveau de gris:

int getGrayScale();

Inversement la couleur peut être spécifié à partir de la valeur RVB:

void setColor(int RVB);

si les couleurs sont non supportes vous pouvez spécifier le niveau de gris:

void setGrayScale(int value);

il n'existe pas de méthode pour spécifier la couleur de fond .Vous devez sélectionner la couleur voulus en fond avec **setcolor();** , puis trace une zone rectangulaire avec la méthode **filRect();** avec en paramètre la taille du Canvas. [09]

Le Style de Trace

Une trace graphique peut avoir un style solide (SOLID) où pointille (DOTTEN). Les caractéristiques du pointille sont spécifique au terminal. [09]

Le Style est specifier par:

void setStrokeStyle(int Style);

Le style en cours peut etre obtenus par:

int getStrokeStyle();

Le Trace de Texte

Il est possible d'afficher du texte en format graphique
Il existe pour cela deux variantes:

- Afficher du texte provenant d'un tableau de caractères .ces méthodes affichent un caractères a la position (x,y) pour l'un ,une partie d'un tableau de caractères a la position (x,y) pour l'autre.

  void drawChar(char character , int x, int y, int anchor);
  void drawChars(char[] data, int offset, int length , int x, int y, int anchor);
- Affiche une chaine String ou une partie de chaine String:

  Void drawString(String str , int x, int y, int anchor);
  Void drawSubstring(String str ,int offset , int len , int x, int y, int anchor);

[29]
Ces méthodes prennent en paramètre le point d'ancrage, ces derniers utilisent pour reduire le calcul nécessaire pour placer le texte. Trois constantes horizontales sont disponible : LEFT , HCENTER et RIGHT , ainsi que quatre constantes verticales , TOP , BASELINE , VCENTER et BOTTOM. [29]
Ces constantes peuvent être combinées avec l'opérateur |.

Ainsi :

g.drawString("core j2me" , x , y , Graphics.BASELINE|Graphics.HCENTRE);
Produit le résultat suivant:

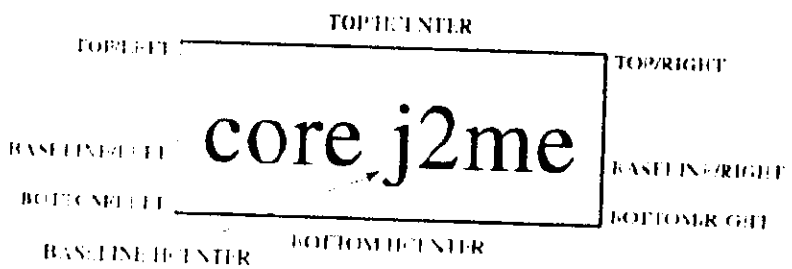


Figure 3-15:Alignement d'un Texte

L'Affichage d'une Image

Une image peut être dessinée grâce a la méthode:

void drawImage(Image img , int x , int y , int anchor);

Elle peur être soit muable , soit immuable .Une image immuable ,contrairement a une image muable ,ne peut être modifier une fois crée .La notion de point d'ancrage pour les images est similaire a celle des texte , excepter que **BASELINE** est remplace par **VCENTER** .[12]

Le Double Tampon

Vous pouvez remarquez parfois un clignotement lors d'affichages répètes .cela est du au fait qu'avant d'afficher l'image suivante, le terminal doit effacer la précédente .En fin d'éviter ce désagrément, vous pouvez utiliser la technique du double–tampon cela revient a utilises un Canvas virtuel supplémentaire et a y préparer l'image suivante. [12]

Concrètement au lieu sur le Canvas, vous commencer par crée une image tampon, qui est une copie du Canvas vous, Au moment du la mise a jour de l'affichage, vous dessiner non plus sur le Canvas mais sur l'image tampon. En fin, cette image est copiée sur le Canvas [12]

Certains terminaux utilisent déjà un tel système. Dans ce cas il set inutile d'en mettre vous-même, vous pouvez savoir si c'est le cas en appelant la méthode:

boolean isDoubleBuffered();

La Classe Font:

La Classe Font représente les polices de caractères ainsi que les métriques associer. Elle ne peut être crée par une application .seul les objets Font préexistantes sur les terminaux peuvent être utilises :

static Font getFont(int face , int style , int size);

La face peut être FACE_MONOSPACE .il s'agit dans ce cas d'une police de caractères monospace .Elle peut aussi être FACE_PROPOSIONAL, c a d une police proportionnelle .En fin, Elle peut être FACE_SYSTEM autrement dit système .

Le style peut être STYLE_BOLD, qui correspond au style grand, STYLE_ITALIC Italique, STYLE_PLAIN normal ,STYLE_UNDERLINED souligner . [13]

En fin, la taille peut être SIZE_LARGE , grande, SIZE_SMALL ,petite et SIZE_MEDIUM ,moyenne . [13]

Ces trois propriétés peuvent par la suite être récupéré:

int getFace();
int getStyle();
int getSize();

Les valeurs des attributs de style peuvent être combinées avec l'opérateur |.

Elles peuvent également être obtenus individuellement par ces méthodes:

boolean isBold();
boolean isItalic();
boolean isPlain();

boolean isUnderlined();

La police de caractères par défaut est retournes par la méthode :
static Font getDefaultFont();

3-5-3-3. Gestion Des Evénements:

La classe Canvas permet d'écrire des applications qui peuvent accéder aux événements de saisie de bas niveau , offrant ainsi un grand control sur l'affichage .Les jeux sont la meilleurs illustrations du type d'applications qui utilisent se mécanisme .
Cette classe est une sous classe de la classe Displayable .Elle permet d'enregistrer un Listener de commandes .Il est cependant nécessaires au préalable de crée plusieurs interface Listener, chacune étant associer a un type d'événement. [13]

Les Méthodes associer aux événements:

A chaque changement d'état d'une MIDlet, , la méthode **showNotify()** est t'appeler avant de rendre visible le Canvas.La méthode **hideNotify()** est appeler après que le Canvas a été retirer du Display .ces changement sont du par exemple a la mise en avant/arrière plan du MIDlet par le gestionnaire d'applications.[25]

Les Touches
Chaque touche a la quelle un événement est associer est identifiée par un code de touche le MIDP définit les touches 0 à 9,* et # dans la case Canvas [25].

le tableau suivant recense les codes des Touches:

Name	*Constant Value*
KEY_NUM0	48
KEY_NUM1	49
KEY_NUM2	50
KEY_NUM3	51
KEY_NUM4	52
KEY_NUM5	53
KEY_NUM6	54
KEY_NUM7	55
KEY_NUM8	56
KEY_NUM9	57
KEY_STAR	42
KEY_POUND	35

Figure 3-16:Liste des Touches

Bien que d'autres touches puissent exister sur certains terminaux, les applications ne doivent utiliser que les touches standard de façon à garantir la portabilité [25]

Chaque touche a un nom, qui peut être obtenu par la méthode:
String getKeyName(int KeyCode);

Il existe trois autres types de méthodes de gestion d'événement relatifs a ses touches :

protected void KeyPressed(int KeyCode);
protected void KeyReleased(int KeyCode);
protected void KeyRepeated(int KeyCode);

L'événement **KeyRepeated(int KeyCode)** n'est pas toujours supportes .Pour savoir s'il l'est appelez la méthode :

Boolean hasRepeatedEvents();

Les Actions de Jeu

Des actions de jeu sont prédéfinies, elles correspondes par exemple, aux flèches de déplacement, le MIDP définit les Actions de jeu recenses au tableau suivant [25]:

Name	*Description*	*Constant Value*
UP	Move up	1
DOWN	Move down	6
LEFT	Move left	2
RIGHT	Move right	5
FIRE	Fire	8
GAME_A	Custom	9
GAME_B	Custom	10
GAME_C	Custom	11
GAME_D	Custom	12

Figure 3-17:Liste des Touches de jeu

Chaque code de jeu est associer a une action de jeu, une action de jeu peut être associer a plusieurs codes de touches .La Translation entre les deux est faite grâces aux méthodes [25] :

GetKeyCode
getGameAction

Si votre application utilise des actions de jeu et que vous vouliez garantir sa portabilité. Vous devez impérativement traduire les événement relatives aux touches en action de jeu UP, DOWN, LEFT et RIGHT . qui peuvent être associer différemment selon le terminal. La Méthode **getGameAction** retourne ainsi l'action de jeu LEFT quand l'utilisateur appuie sur le touche se trouvant physiquement sur la gauche du terminal.
Comme avec les sous classes de la classe Displayable, vous pouvez ajouter des commandes a un objet Canvas et enregistrer un **CommandListener** au près de l'objet Canvas. [25]

3-6. Conclusion

Comme nous avons vu dans le chapitre précèdent .Nous pensons que la plate forme j2me est parfaitement adaptée pour devenir le langage standard de développement des jeux destinés aux appareils sans fil. Maintenant nous présentons les liens existants entre cette plate forme et la communication en réseau sans fil, avec l'introduction de la technologie Bluetooth.

Chapitre 4 :

J2ME & Réseau sans Fils (Bluetooth)

4- J2ME & Réseau sans Fils (Bluetooth)

4-1. Introduction

Suite au développement croissant des nouvelles technologies, on assiste a une nouvelle vague de produits offerts sur le marche. La majorité de ces produits s'inscrivent dans le cadre d'une technologie sans-fil. Plus besoin de câbles ou de connectique pour que des appareils communiquent et interagissent entre eux. Non seulement ces derniers arrivent à communiquer, mais aussi sur des distances de plus en plus allongées. Cette nouvelle vision a bel est bien marqué les esprits, ce qui se justifie par le déclenchement d'une nouvelle aire: **Un monde Sans-fil**. Le sans-fil est considère par les acteurs de la haute technologie comme la nouvelle poule aux oeufs d'or. La preuve, outre le Wi-Fi et le Bluetooth, qui ont atteint une certaine reconnaissance, on assiste à la multiplication des normes : **ZigBee** à **Wimax**, chacune étant soutenue par un consortium d'entreprises différent.

4-2. Bluetooth

4-2-1. Présentation

Bluetooth, est une technologie de réseau personnel sans fils (note WPAN pour Wireless Personal Area Network), c'est-à-dire une technologie de réseaux sans fils d'une faible portée permettant de relier des appareils entre eux sans liaison filaire. Contrairement a la technologie IrDa (liaison infrarouge), les appareils Bluetooth, ne nécessitent pas une ligne de vue directe pour communiquer, ce qui rend plus souple son utilisation et permet notamment une communication d'une pièce a une autre, sur de petits espaces. Elle a été conçue dans le but de remplacer les câbles entre les ordinateurs et les imprimantes, les scanners, les souris, les téléphones portables, les PDAs et les appareils photos numériques. [23]

4-2-2. Historique

Son nom est directement inspiré du roi danois Harald II surnomme Harald II Blatand (a la dent bleue), connu pour avoir réussi a unifier les états du Danemark, de Norvège et de Suède. Le Réseau local sans fil est ressemant utilisé par une seule personne donc c'est un réseau personnel. D'une portée de quelques dizaines de mètres Le logo de Bluetooth, est d'ailleurs compose des runes nordiques H et B.

- 1994 : Création par le fabricant suédois Ericsson.
- Février 1998 : Plusieurs grandes sociétés (Agere, IBM, Intel, Microsoft, Motorola, Nokia et Toshiba) s'associent pour former le Bluetooth, Special Interest Group (SIG)
- Juillet 1999 : Sortie de la spécification 1.0

L'objectif de Bluetooth, est de permettre de transmettre des données ou de la voix entre des équipements possédant un circuit radio de faible coût, sur un rayon de l'ordre d'une dizaine de mètre à un peut moins d'une centaine de mètre et avec une faible consommation électrique.

Ainsi, la technologie Bluetooth, est principalement prévue pour relier des périphériques (imprimantes, téléphones portables, appareils domestiques, oreillettes

sans fils, souris, clavier, etc.), des ordinateurs ou des assistants personnels (PDA), sans utiliser de liaison filaire. La technologie Bluetooth, est également de plus en plus utilisée dans les téléphones portables, afin de leur permettre de communiquer avec des ordinateurs ou des assistants personnels et surtout avec des dispositifs mains-libres tels que des oreillettes Bluetooth, Les oreillettes Bluetooth, permettent de faire office de casque audio perfectionné intégrant des fonctionnalités de commande a distance. [23]

4-2-3. Caractéristiques

Le Bluetooth, permet d'obtenir des débits de l'ordre de 1 Mbps, correspondant a 1600 échanges par seconde en full duplex, avec une portée d'une dizaine de mètres environ avec un émetteur de classe II et d'un peut moins d'une centaine de mètres avec un émetteur de classe I.
Contrairement a la technologie IrDa, principale technologie concurrente utilisant des rayons lumineux pour les transmissions de données, la technologie Bluetooth, utilise les ondes radio (bande de fréquence des 2.4 GHz) pour communiquer, si bien que les périphériques ne doivent pas nécessairement être en liaison visuelle pour communiquer. Ainsi deux périphériques peuvent communiquer en étant situes de part et d'autre d'une cloison et, cerise sur le gâteau, les périphériques Bluetooth, sont capables de se détecter sans intervention de la part de l'utilisateur pour peu qu'ils soient a portée l'un de l'autre. [23]

La Pile de Protocoles

Afin d'assurer une compatibilité entre tous les périphériques Bluetooth,, la majeure partie de la pile de protocoles est définie dans la spécification. Les éléments fondamentaux d'un produit Bluetooth, sont définis dans les deux premières couches protocolaires, la couche radio et la couche bande de base. Ces couches prennent en charge les taches matérielles comme le contrôle du saut de fréquence et la synchronisation des horloges. [19]

La Couche Radio (RF)

La couche radio (la couche la plus basse) est gérée au niveau matériel. C'est elle qui s'occupe de l'émission et de la réception des ondes radio. Elle définit les caractéristiques telles que la bande de fréquence et l'arrangement des canaux, les caractéristiques du transmetteur, de la modulation, du receveur, etc...

La technologie Bluetooth utilise l'une des bandes de fréquences ISM (Industrial, Scientific et Medical) réserve pour l'industrie, la science et la médecine. La bande de fréquences utilisée est disponible au niveau mondial et s'étend sur 83,5 MHz (de 2,4a 2,4835 GHz).
Cette bande est divisée en 79 canaux sépares de 1 MHz.
Le codage de l'information se fait par sauts de fréquence.
La période est de 625 micro s ce qui permet 1600 sauts par seconde.

Le standard Bluetooth, définit en effet 3 classes d'émetteurs proposant des portées différentes en fonction de leur puissance d'émission :
Classe Puissance (affaiblissement)
Portée

I-100 m W(20dBm) 100 m
II-2,5 m W(4dBm) 15-20 m
III-1 m W (0 dBm) 10 m

La plupart des fabriquant d'appareils électroniques utilisent des modules de classe 3.
La bande de base (baseband)
La bande de base (ou baseband en anglais) est également gérée au niveau matériel.

C'est au niveau de la bande de base que sont définies les adresses matérielles des périphériques (équivalent a l'adresse MAC d'une carte réseau). Cette adresse est nommée BD ADDR (Bluetooth Device Address) et est codée sur 48 bits. Ces adresses sont gérées par la IEEE Registration Authority. C'est également la bande de base qui gère les différents types de communication entre les appareils. Les connexions établies entre deux appareils Bluetooth,peuvent être synchrones ou asynchrones.
La bande de base peut donc gérer deux types de paquets :
– Les paquets SCO (Synchronous Connection-Orientated).
– Les paquets ACL (Asynchronous Connection-Less). [19]

Réseau piconet

Un piconet est un mini réseau qui se crée de manière instantanée et automatique quand plusieurs périphériques Bluetooth, sont dans un même rayon. Un piconet suit une topologie en étoile : 1 maître avec plusieurs esclaves.
Un périphérique maître peut administrer jusqu'á :
– 7 esclaves actifs
– 255 esclaves en mode parked

La communication est directe entre le maître et un esclave. Les esclaves ne peuvent pas communiquer entre eux. Tous les esclaves du piconet sont synchronises sur l'horloge du maître. C'est le maître qui détermine la fréquence de saut pour tout le piconet. [19]

Réseau scatternet

Les périphériques esclaves peuvent avoir plusieurs maîtres, les différents piconets peuvent donc être relies entre eux. Le réseau ainsi forme est appelée un scatternet (litteralement reseau chaine). [19]

Le controlleur de liaisons (LC)

Cette couche gère la configuration et le contrôle de la liaison physique entre deux appareils.

Le gestionnaire de liaison (LM)

Cette couche gère les liens entre les périphériques maîtres et esclaves ainsi que les types de liaisons (synchrones ou asynchrones).
C'est le gestionnaire de liaisons qui implémente les mécanismes de sécurité comme :
– l'authentification,
– le pairage,

– la création et la modification des clés,
– et le cryptage.
L'interface de contrôle de l'hôte (HCI)
Cette couche fournit une méthode uniforme pour accéder aux couches matérielles. Son rôle de séparation permet un développement indépendant du hardware et du software.Les protocoles de transport suivants sont supportes :
– USB (Universal Serial Bus)
– PC-Card
– RS-232
– UART
La couche L2CAP
L2CAP signifie Logical Link Control et Adaptation Protocol.
Multiplexage

Cette couche permet d'utiliser simultanément différents protocoles de niveaux supérieurs. Un mécanisme permet d'identifier le protocole de chaque paquet envoyer pour permettra l'appareil distant de passer le paquet au bon protocole, une fois celui-ci récupère. [19]

Segmentation et réassemblage

La couche L2CAP gère également la segmentation (et le réassemblage) des paquets de protocoles de niveaux supérieurs en paquets de liaison de 64 Ko.
Les Services
–RFCOMM: est un service base sur les spécifications RS-232, qui émule des liaisons séries,Il peut notamment servira faire passer une connexion IP par Bluetooth,.
–SDP: SDP signifie Service Discovery Protocol.
Ce protocole permettra un appareil Bluetooth de rechercher d'autres appareils et d'identifier les services disponibles. Il s'agit d'un élément particulièrement complexe de Bluetooth,.
–OBEX: OBEX signifie Objet Exchange. Ce service permet de transférer des données grâce a OBEX, protocole d'échange de fichiers IrDa. [19]

Les Profils

Un profil correspond à une spécification fonctionnelle d'un usage particulier. Ils peuvent également correspondre à différents types de périphériques.
Les profils ont pour but d'assurer une interopérabilité entre tous les appareils Bluetooth, Ils définissent :

– La manière d'implémenter un usage défini.
– Les protocoles spécifiques a utilisé
– Les contraintes et les intervalles de valeurs de ces protocoles

Les différents profils sont :
1. GAP : Generic Access Profile
2. SDAP : Service Discovery Application Profile
3. SPP : Serial Port Profile
4. HS Profile : Headset Profile
5. DUN Profile : Dial-up Networking Profile

6. LAN Access Profile
7. Fax Profile
8. GOEP : Generic Object Exchange Profile
9. SP : Synchronization Profile
10. OPP : Object Push Profile
11. FTP : File Transfer Profile
12. CTP : Cordless Telephony Profile
13. IP : Intercom Profile

Le profil d'accès générique (GAP)

Le profil d'accès générique est le profil de base dont tous les autres profils héritent. Il définit les procédures génériques de recherche d'appareils, de connexion et de sécurité. [19]

4-2-4. Les Normes

Le standard Bluetooth se décompose en différentes normes :

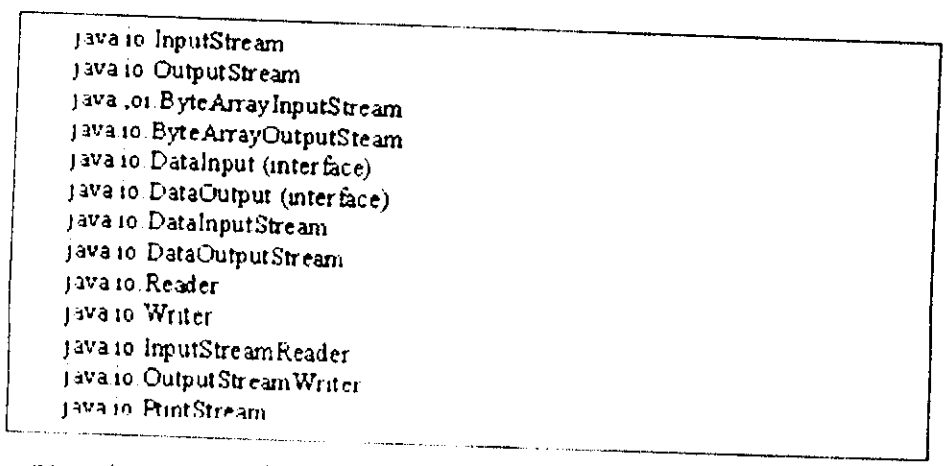
- IEEE 802.15.1 définit le standard Bluetooth 1.x permettant d'obtenir un débit de 1 Mbit/s.
- IEEE 802.15.2 propose des recommandations pour l'utilisation de la bande de fréquence 2.4 GHz (fréquence utilisée également par le Wi-Fi). Ce standard n'est toute fois pas encore valide.
- IEEE 802.15.3 est un standard en cours de développement visant a proposer du haut débit (20 Mbit/s) avec la technologie Bluetooth
- IEEE 802.15.4 est un standard en cours de développement pour des applications Bluetooth à bas débit. [19]

4-3. Le GCF (The Generic Connection Framework)

4-3-1. Présentation

Java.io.* et java.net.* sont des paquets de J2SE qui ne conviennent pas pour les mobiles à cause de leurs ressources limitées. C'est la raison pour laquelle un nouveau paquet java.microedition.io a été développé.

I/O classes:



Ce paquet ne contient qu'une seule classe : **Connector**, 8 interfaces et **ConectionNotFoundException**.

La classe Connector est le " coeur " de Connection Framework. Elle renferme une série de méthodes statiques pour la réception de l'objet Connection. Si l'opération réussie, la méthode rend l'objet qui réalise une connexion à l'interface, ou bien, IOException s'apparaît.

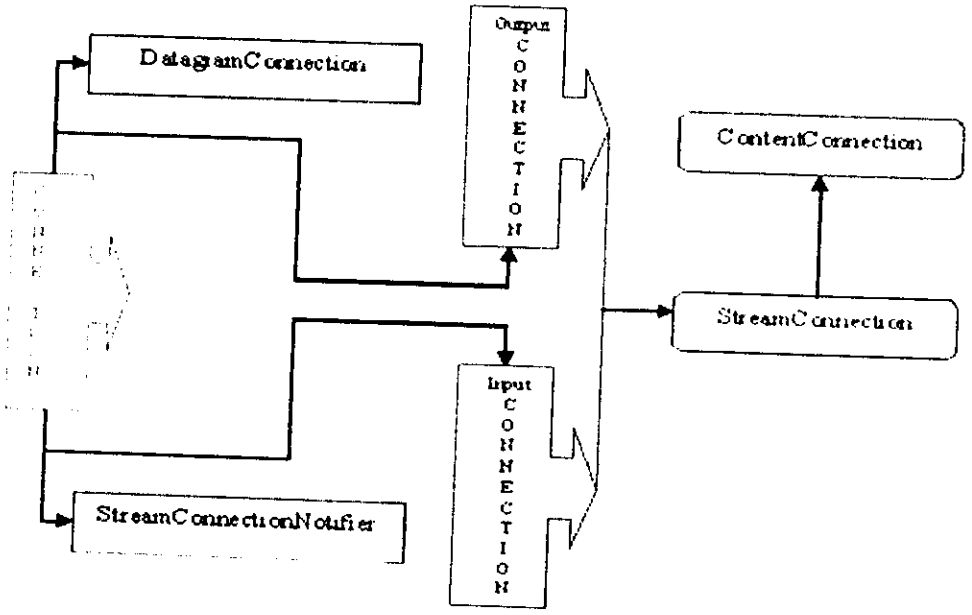


Figure 4-1:La hiérarchie des interfaces

L'objet réalisant la connexion à l'interface peut être reçu grâce à la classe Connector comme on l'avait dit ci-dessus. L'interface Connection renferme la seule méthode close() qui peut fermer la connexion au réseau, voici les autres interfaces:

-InputConnection est l'interface qui permet de lire les données. Les méthodes **openInputStream** et **openDataInputStream** retournent les données pour la lecture

-OutputConnection, est l'interface qui permet d'enregistrer les données. Les méthodes **openOutputStream** et **openDataOutputStream** retournent les données pour l'enregistrement

-StreamConnection, est l'interface qui réunie **InputConnection** et **OutputConnection**.

-ContentConnection est la sous interface de **StreamConnection**

-SteamConnectionNotified attend que la connexion soit établie. La méthode **acceptAndOpen()** rend le **StreamConnection** objet ce cette connection.

-DatagrameConnection interface pour la connexion en datagramme

-ConnectionNotFoundException une exception qui s'apparaît quand la connexion ne peut pas être établie.

4-3-2. Exemple de Connexion

Le GCF fournit un moyen uniforme et pratique d'effectuer des entrées / sorties quelque soit le type de protocole [17]. La syntaxe générale est la suivante :

Connector.open("<protocole>://<adresse>:<parametres>");

Plusieurs protocoles sont utilises. Nous allons voir les plus important:

Vous pouvez par exemple lire un fichier en passant par le système de gestion de fichier du terminal:
Connector.open ("file://monfichier.txt");

Le plus fréquent sera probablement d'utiliser le célèbre protocole HTTP. Ainsi, vous pouvez télécharger la page d'accueil du site Yahoo :
Connector.open ("http://www.yahoo.fr");

Si vous voulez créer une application de type peer 2 peer, vous pourez utiliser les sockets :
Connector.open ("socket://www.monsite.com:8001");

Vous pouvez également contrôler l'éventuel port série du terminal :
Connector.open("comm://9600:18N");

4-4. Bluetooth & GCF (JSR 082)

4-4-1. Présentation

Les profils de Bluetooth - ne pas confondre avec des profils de J2ME - décrivent les conditions d'interopérabilité et d'uniformité de cette plateforme. Ils incluent *le Generic Access Profile* qui définit les fonctionnalités du terminal, le *Service Discover Application Profile* qui définit les aspects des services découverte, et le *Serial Port Profile* qui définit le port série du terminal. [03]

La pile de Bluetooth comporte une pile de logiciel qui se connecte à une interface hardware, comme le schéma illustre :

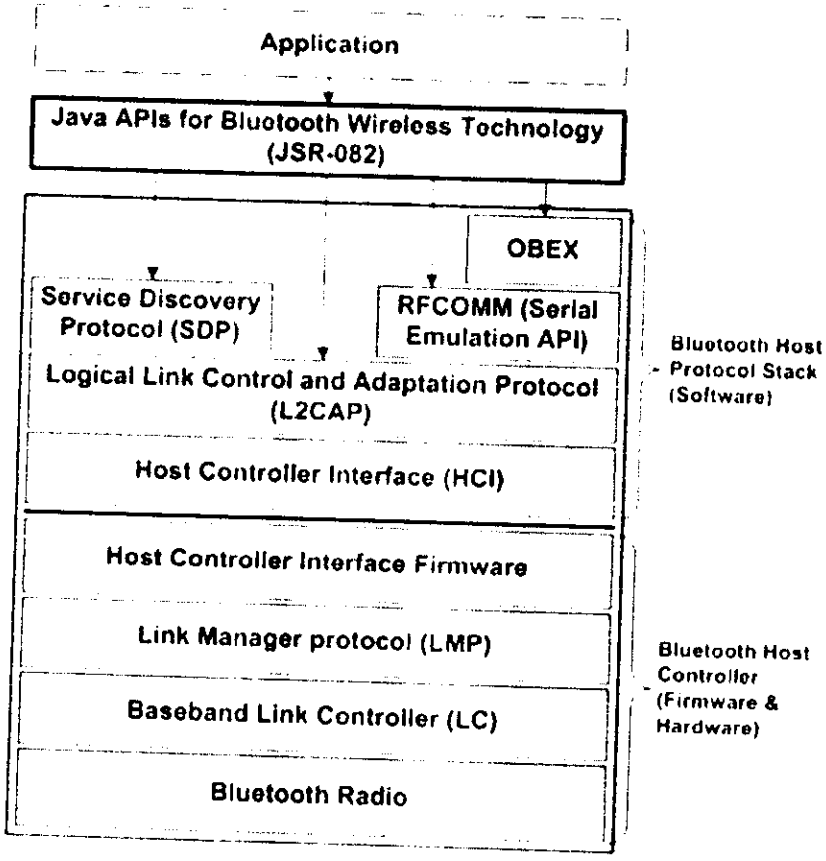


Figure 4-2:La Pile du Protocol Bluetooth

JSR 82 présente la pile de logiciel de Bluetooth aux développeurs de plateforme Java. Avec un intérêt spécial au Service Discovery Protocol (SDP),le Serial Port Profile RFCOMM pour l'émulation du port série, et Le Logical Link Control et Adaptation Profile (L2CAP) qui fournit des services de connection aux protocoles de tels que l'opération de segmentation , remontage, et le multiplexage.

JABWT inclut également l'APIs d'objet. OBEX qui permet d'échanger des *objets* et fichiers entre terminaux. [03]

Une application Bluetooth peut fonctionner soit en mode serveur ou client - un producteur des services ou un consommateur de services - ou elle peut se comporter comme véritable point peer-2-peer en exposant ses services a ses clients. Le schéma suivant illustre le diagramme d'utilisation pour une application Bluetooth [03]:

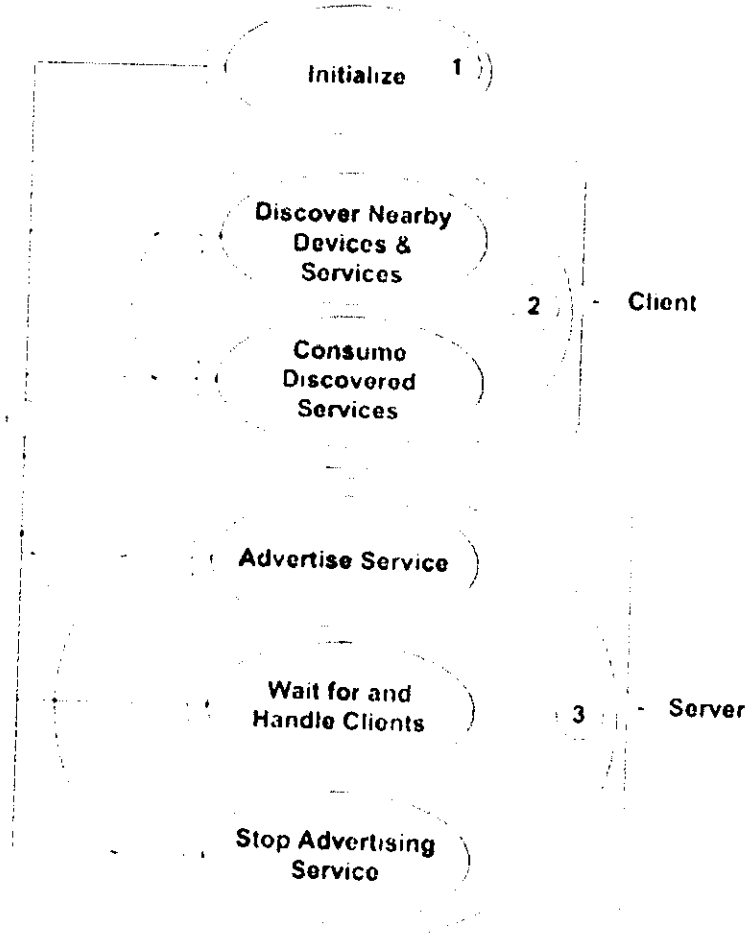


Figure 4-3:Diagramme d'utilisation pour une application Bluetooth

Ces cas d'utilisation peuvent être récapitulés ainsi :

- **Initialisation** - toutes les applications Bluetooth doivent d'abord initialiser la pile Bluetooth.
- **Client** - un client consomme les services à distance. Il recherches d'abord tous les terminaux Bluetooth voisins, puis à chaque terminal découvert les services d'intérêt pour elle.
- **Serveur** - le serveur rend ses services disponibles aux clients. Il les enregistrent dans la base de données de découverte de service (SDDB). Il attend alors les connexions des clients entrantes, les acceptent et ils les servent. En conclusion, les services disponibles sont tous enregistrés dans la SDDB.

Voici Le schéma de diagramme d'activité des applications Bluetooth [03]:

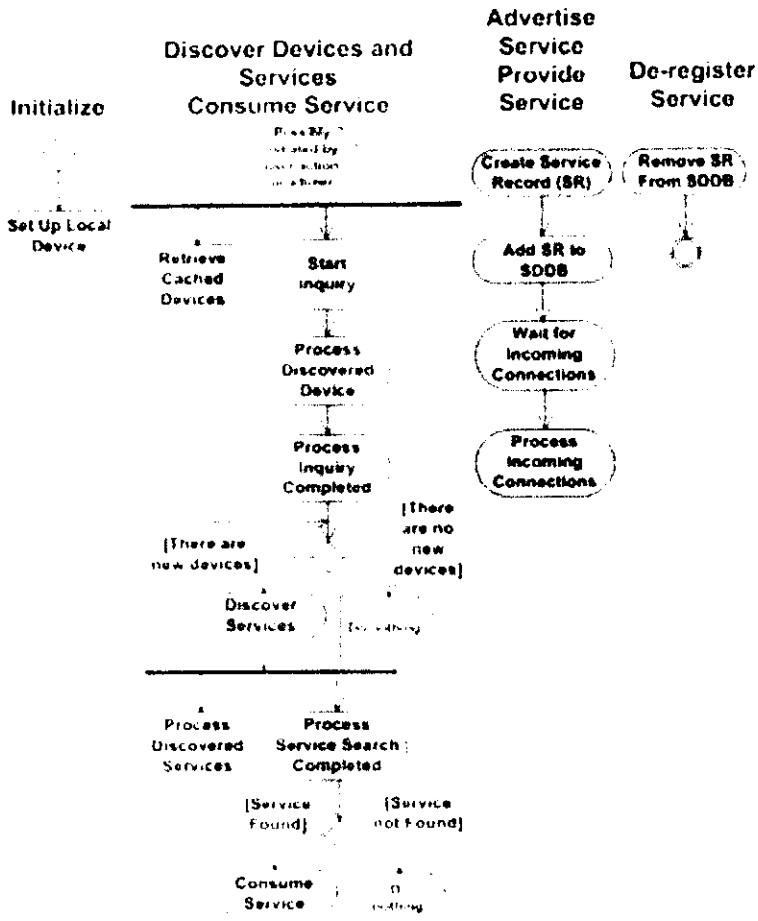


Figure 4-4:Diagramme d'activité pour une application Bluetooth

Le schéma suivant illustre les classes et interfaces utilisées par une MIDlet implémentant le Bluetooth. [03]:

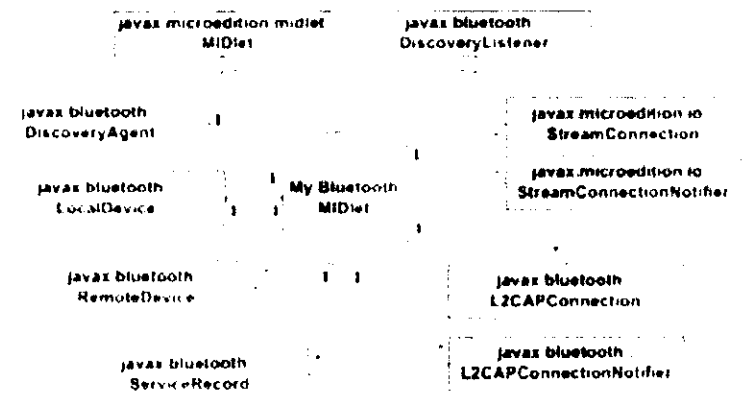


Figure 4-5:les Interfaces Bluetooth-MIDlet

Nous pouvons décomposer le noyau Bluetooth par rapport javax.bluetooth, dans trois catégories: découverte, gestion des terminaux, et communication.

4-4-2. Les Bluetooth Discovery APIs

Les applications client emploient les APIs de découvertes *de* JABWT pour chercher les terminaux et ses services. La classe **DiscoveryAgent** supporte la découverte des terminaux et des services. Les applications cliente doivent implémenter et enregistrer l'interface **DiscoveryListener**, qui définit les appels pour le terminal et les notifications pour les services découvert. [03]
Voici la relation entre une application Bluetooth cliente et **DiscoveryAgent**

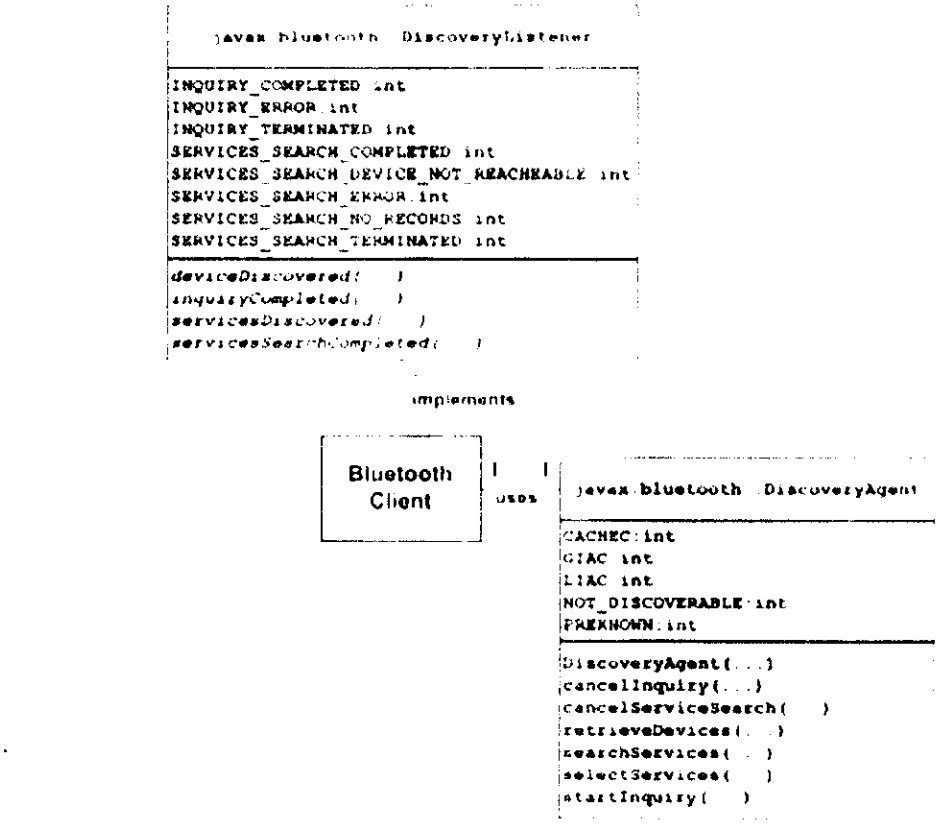


Figure 4-6:la classe DiscoveryAgent et l'Interface DiscoveryListener

4-4-2-1. APIs pour Device Discovery

DiscoveryAgent est utilisé pour découvrir et rechercher les terminaux , avec l'aide des méthodes suivantes :

retrieveDevices():recherche les terminaux déjà découverts.
startInquiry(): lance la recherche des terminaux
cancelInquiry():sort de n'importe quelle recherche actuellement en marche.

Le Bluetooth Discovery Agent appel le **DiscoveryListener** du terminal en différents points du processus de découverte en utilisant les fonctions suivantes:

deviceDiscovered() indique si le terminal est **Discovered**.
inquiryCompleted() indique si la recherche a abouti a un succès, a une erreur, ou a été annulé.

Le diagramme d'état suivant illustre les états du **device-discovery** après les appels du **DiscoveryListener**:

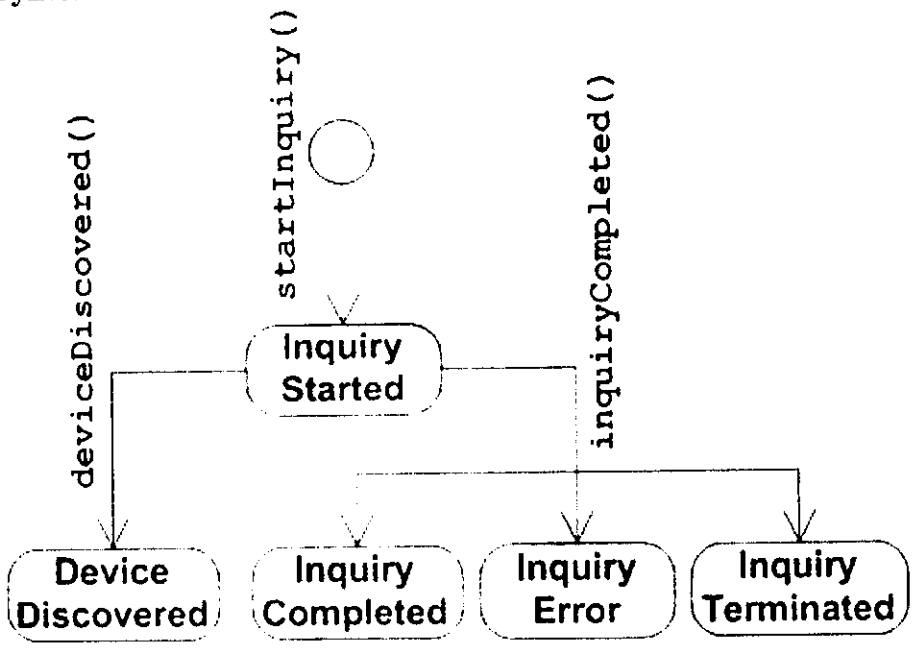


Figure 4-7:Diagramme d'Etats de la recherche des Terminaux

La découverte débute par l'appel du **startInquiry()**, après progression de la recherche le **Bluetooth Discovery Agent** invoque **deviceDiscovered**() et **inquiryCompleted**(). [03]

4-4-2-2. APIs pour le Service Discovery:

Ici aussi **DiscoveryAgent** est utilisé pour lancer la découverte des services avec les méthodes suivantes:

selectService() essaie de sélectionner un service.
searchServices() commence la découverte des services
cancelServiceSearch() annule la découverte des services en cours.

Le Bluetooth Discovery Agent appel le **DiscoveryListener** du terminal en différents points du processus de découverte pour la recherche des services grâces aux fonctions suivantes:
servicesDiscovered()indique si le service dans le terminal est **Discovered**.
serviceSearchCompleted()indique si la recherche des services est finis.

Le diagramme d'état suivant illustre les états du **service-discovery** après les appels du **DiscoveryListener**:

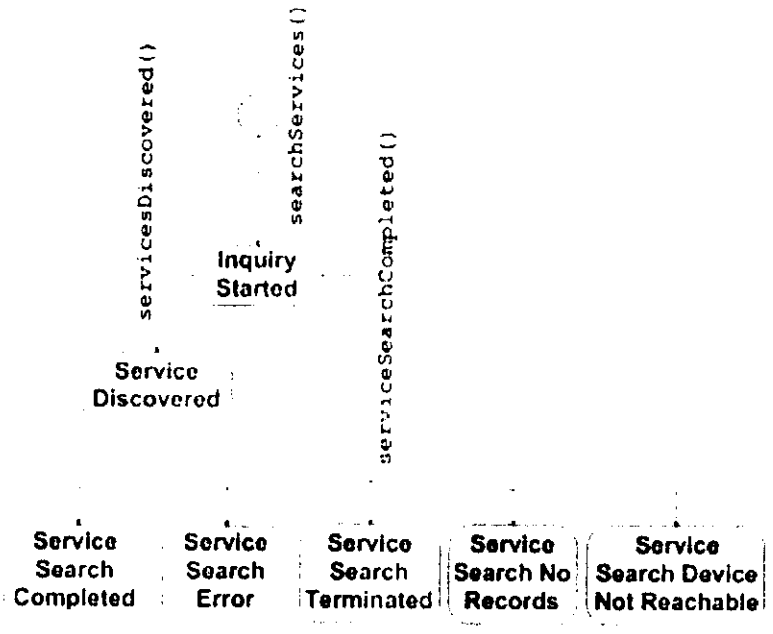


Figure 4-8:Diagramme d'Etats de la recherche des Servises

La découverte des services débute par un appel a **searchServices()** après progression de la découverte le **Bluetooth Discovery Agent** invoque **servicesDiscovered()** et **serviceSearchCompleted().**

En plus de **DiscoveryAgent** et de **DiscoveryListener**, vous pouvez utiliser également les classes **UUID**, **ServiceRecord,** et **DataElement** pour la recherche des services. [03]

4-4-3. La classe d'UUID

Dans Bluetooth, chaque service et attribut de service est uniquement identifié par un Universally Unique Identifier (UUID). Comme son nom le suggère, chaque terminal Bluetooth a un "UUID" unique. La classe d'UUID peut être représenté sous un entier short(16 - ou de 32 bits) ou bien long (128-bit). Elle fournit des constructeurs pour créer un UUID d'une longueur de 16 ou 32 bits, une autre méthode pour comparer deux "UUID", et une dernière méthode pour convertir un UUID sous forme d'un **String**. Les instances d'UUID sont constantes par appareil, et seulement les services identifiés par UUID sont accessibles.

Pour produire un UUID on utilise la commande : **uuidgen - t** sous Linux ou bien **uuidgen** sous Windows. [18]

L'UUID est de forme suivante: 2d266186-01fb-47c2-8d9f-10b8ec891363

4-4-2-4. La SDDB et l'Interface ServiceRecord

Au centre du service Discovery il y a la Base de données de Découverte de Service (**SDDB**) et le Protocole de Découverte de Service (SDP). Le SDDB est une base de données dans l'implémentation Bluetooth qui contient *des dossiers de service* disponibles pour les clients. [18]

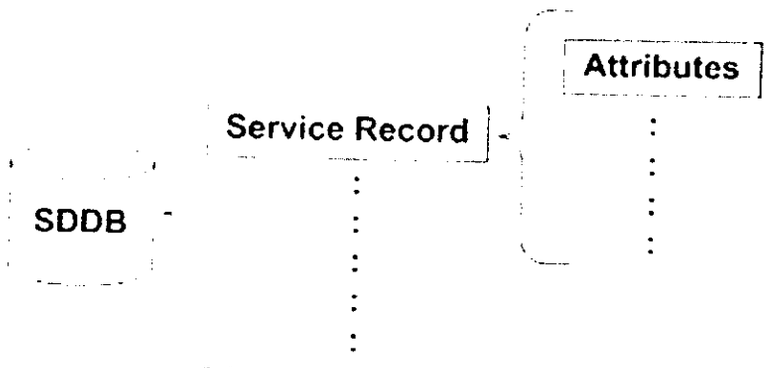


Figure 4-9:La SDDB

Chaque record de service est représenté par un champ de **ServiceRecord.** Ce record contient des attributs qui décrivent le service en détail. La classe fournit plusieurs méthodes utiles voici quelques exemples:

getAttributeIDs () et **getAttributeValue** () récupèrent les attributs de record de service.
getConnectionURL () reçoit la connexion URL pour le serveur accueillant le record de service.
getHostDevice () retourne le **RemoteDevice** accueillant ce service.
populateRecord () et **setAttributeValue()** modifie les attributs du record service.
setDeviceServiceClasses() mis a jour les classes du service.

La figure suivante montre les relations entre les terminaux Bluetooth locaux et lointains, le SDDB et les dossiers de service :

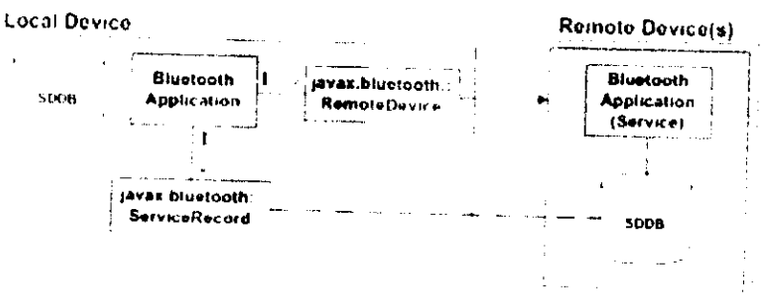


Figure 4-10:La Recherche des Services en utilisant RemoteDevice SDDB, et ServiceRecord

Pour rendre un service disponible pour les clients, une application de serveur crée un record de service en ajoutant une nouvelle connexion,puis insère ensuite, le record dans son SDDB en invoquant la méthode **acceptAndWait()**. Les applications du serveur peuvent récupérer et actualiser le record de service, tandis que les applications de client font appelle a une la SDDB lointaine pour rechercher ses services disponibles. [03]

4-4-5. La Classe DataElement

Un service peut avoir beaucoup d'attributs, certains obligatoires, d'autres optionnels. Un attribut de service est représenté par un objet DataElement, qui fournit des méthodes de mise à jour de ces valeurs d'attribut. [03]

Les attributs obligatoires sont mis automatiquement quand un service est enregistré. Ceux-ci incluent **ServiceRecordHandle**, **ServiceClassIDList**, **ServiceRecordState**, **ServiceID** et **ProtocolDescriptorList**.

Beaucoup d'attributs optionnels sont disponibles, mais trois sont d'intérêt spécial : **ServiceName**, **ServiceDescription** et **ProviderName**.

4-4-6. La Communication Bluetooth

Les connexions de JABWT (Java API Bluetooth Wireless Technolology) sont fondées sur le **Logical Link Control and Adaptation Layer Protocol**. L2CAP est un protocole de bas niveau pour diriger des paquets de données jusqu'à 64 kilo-octets de longeur. L2CAP manipule des détails comme la segmentation de message et le réassemblage (SAR) et la connexion avec multiplexage. En plus, il y a le Profil de Port Série (SPP)et le RFCOMM, un protocole d'émulation du port série sur L2CAP. [03]

L2CAP et les connexions RFCOMM sont fondés sur le Generic Connection Framework (GCF), une hiérarchie directe d'interfaces et de classes pour créer des connexions et exécuter les E/S. JABWT étend GCF en ajoutant L2CAP et RFCOMM voila une figure qui illustre tout ça:

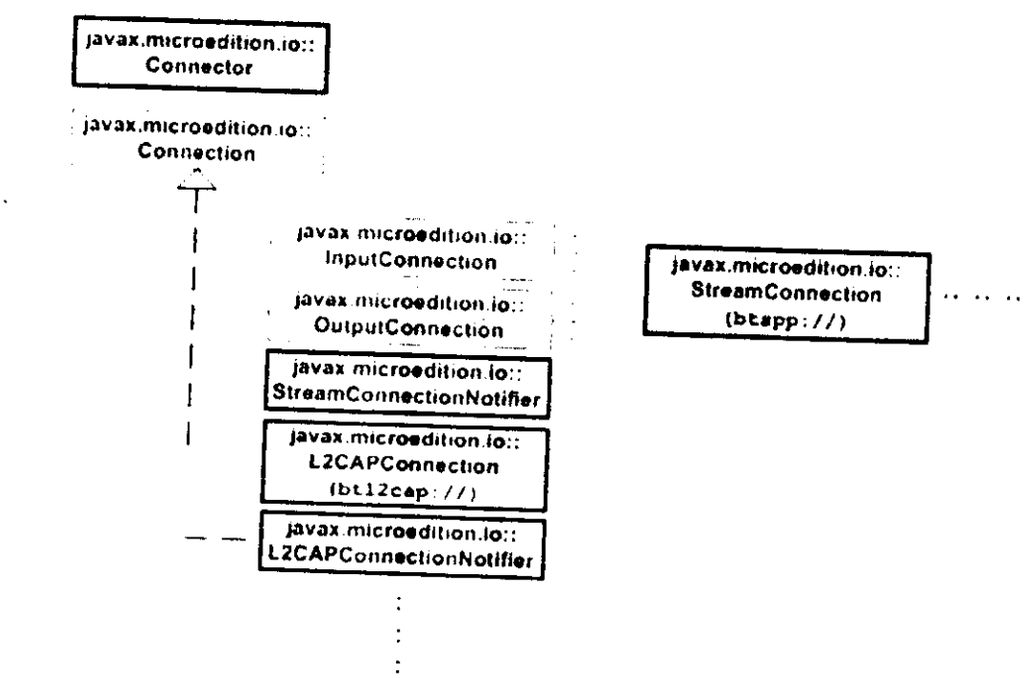


Figure 4-11: Les Types de Connection avec la GCF et Bluetooth

Comme avec tous les types de connexion GCF, vous créez des connexions Bluetooth en appelant **javax.microedition.io.Connector** L'URL de connection passe au connector détermine le type de connexion [03]:

Le Format du L'URL pour une connection L2CAP :
btl2cap://*hostname*:[*PSM* | *UUID*];*parameters*

Le Format du L'URL pour une connection RFCOMMStreamConnection:
btspp://*hostname*:[*CN* | *UUID*];*parameters*

ou:

btl2cap c'est L'URL du type L2CAPConnection de connection.

btspp c'est L'URL du type RFCOMM de connection.

hostname: est le localhost en cas de serveur de connection, ou bien l' adresse Bluetooth pour crée une connection cliente.

PSM est la valeur de multiplixage Protocole/Service, elle est aussi utilisé par les clients comme un port TCP/IP.

CN est la valeur du canal utilisé par les clients pour se connecter au serveur .

UUID c'est la valeur utilisée en installant un service sur un serveur.

parameters incluent le nom pour décrire le nom de service, et les paramètres de sécurité authentification, autorisation, et chiffrement.

Le nom d'hôte dans l'URL de connection indique s'il on a crée une connection cliente ou serveur. En utilisant le **localhost** comme nom d'hôte vas indiquer une connection serveur. Un client souhaitant se relier à un service donné peut rechercher l'URL de la connexion du service en appelant **ServiceRecord.getConnectionURL** ().

4-4-7. Les Exceptions

L'API de **javax.bluetooth** définit trois classes d'exception [18]:

BluetoothConnectionException est lancé quand un Bluetooth L2CAP, RFCOMM, ou une connexion OBEX-over-RFCOMM ne peut pas être établi avec succès.

BluetoothStateException est lancée dans le cas ou la connection est indisponible.

ServiceRegistrationException est lancé quand il y a un échec d'ajout ou un changement d'un record de service dans la Local Service Discovery Database.

4-4-8. Conclusion

Bluetooth est une technologie sans fil passionnante, pour les réseaux personnels, qui permettent aux terminaux mobiles de se reliés automatiquement, de partagés des données, et d'assurés des services. Et c'est le meilleur moyen actuelle pour implémenter des jeux multi joueurs sur des Terminaux Mobiles.

5- Conception & Implémentation

Notre jeu est décomposé en 4 parties qui interagissent entre elles :
- La première partie s'occupe de l'interface haut niveau dédiée aux utilisateurs.
- Une deuxième partie qui représente le jeu lui-même implémentée grâce à l'interface de bas niveau du J2ME.
- Une Troisième partie dédiée à l'intelligence du jeu autrement dit le System Expert a base de Règles.
- Quatrième et dernière partie qui gère les communications réseau par Bluetooth.

On détaillera ces parties dans les sections à venir, tout d'abord, en commence par une description générale de l'organisation de l'interface utilisateur :

5-1. Conception Général de L'Interface Utilisateur

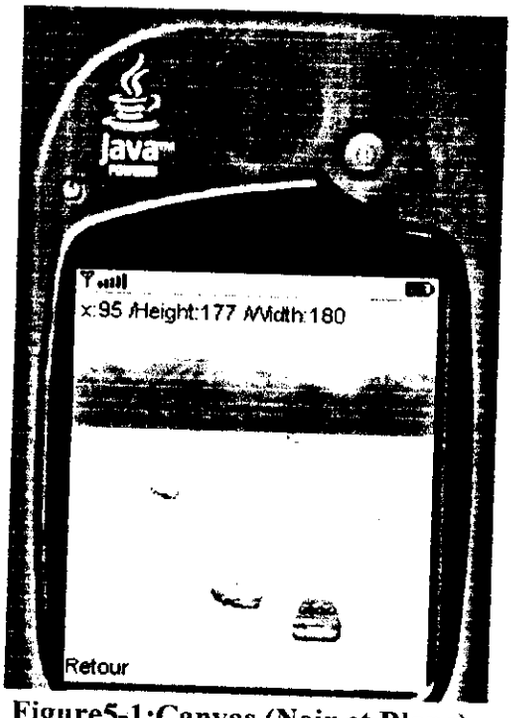


Figure5-1:Canvas (Noir et Blanc)

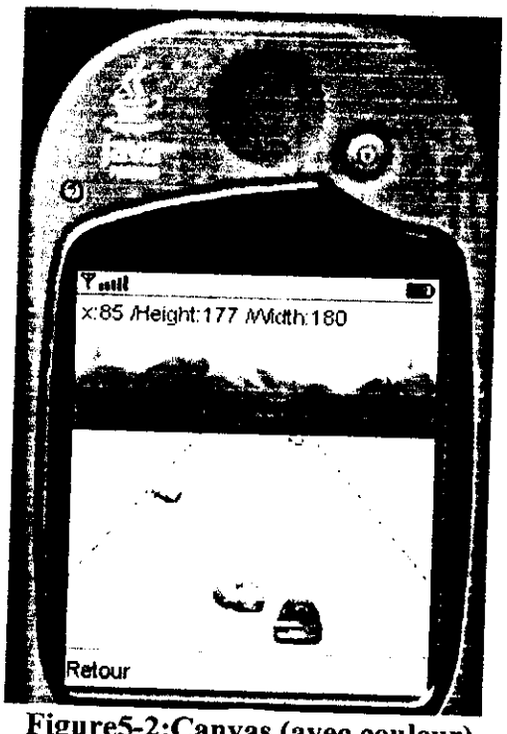


Figure5-2:Canvas (avec couleur)

L'interface utilisateur du MIDlet **SmartRace2** s'adapte à la taille d'écran du téléphone et supporte soit les couleur ou bien le noir et blanc (**grayscale**). La MIDlet fonctionne dans les consoles qui supporte le profil MIDP1.0 standard et le CLDC 1.0. C'est pratiquement touts les appareils qui sont équipés d'une JVM (java virtual machine).

Notre interface utilisateur est organisée tel que montre la figure ci-dessous:

5-1-1. Figure des Screen

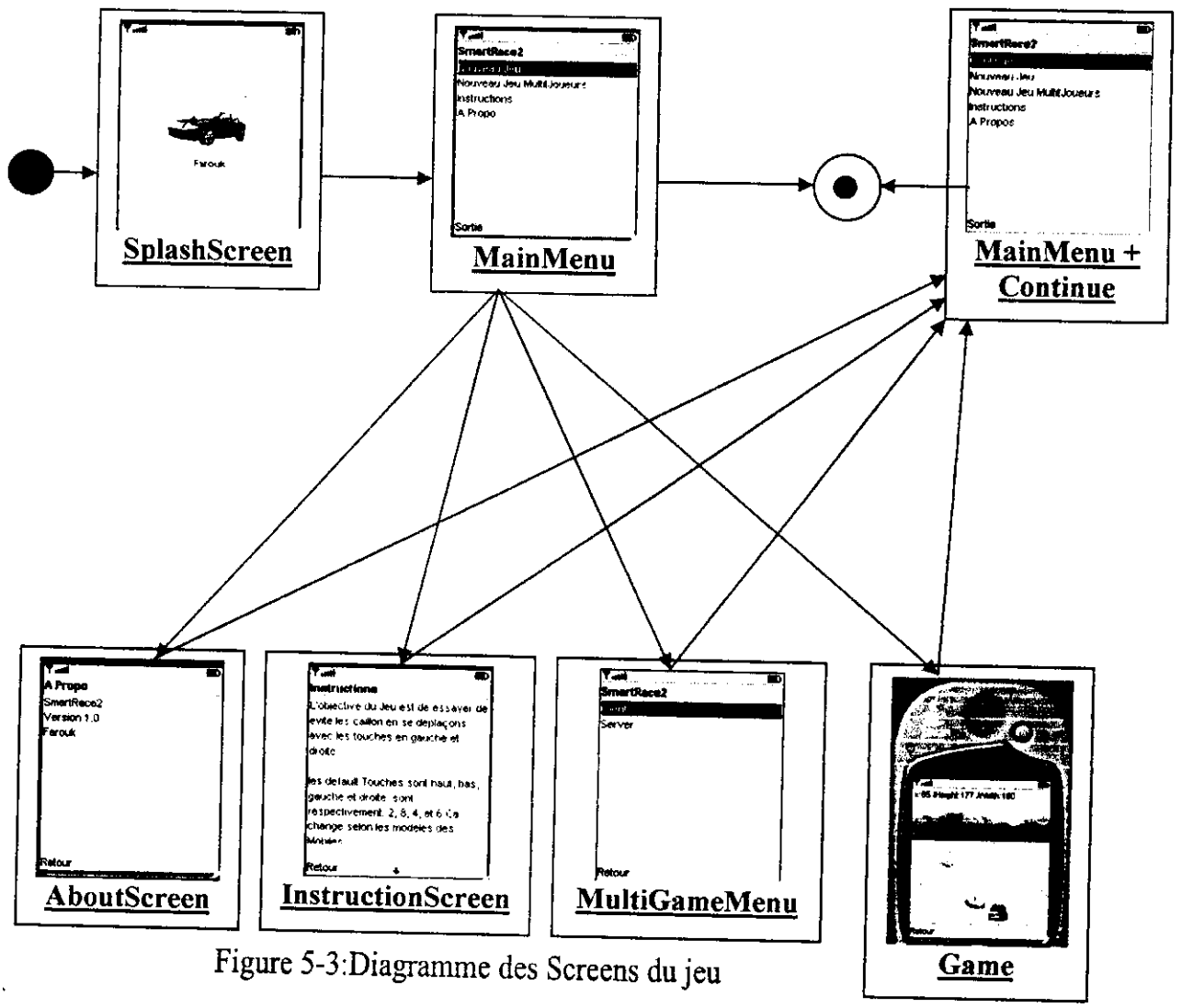


Figure 5-3:Diagramme des Screens du jeu

Comme vous voyer notre interface est organisé en 7 Screen chaque Screen est affecter a une tache bien précise, dans ce que suit, on détaillera les fonctionnalité de chaque Screen :

5-1-2. Le SplashScreen

Son rôle est uniquement esthétique, il est implémentes afin de faire un premier aperçu de notre jeu. C'est un jeu de course de voitures alors il est nécessaire d'afficher une photos d'une voiture, avec le mot anglais "RACE" qui veut dire voiture de course et le nom du constructeur en dessous (qui est le mien).ce Screen va disparaître soit par l'action de l'utilisateur en appuyant n'importe quelle touche, soit automatiquement après 4 secondes, en laissant sa place au **MainMenu** le menu principal du jeu.

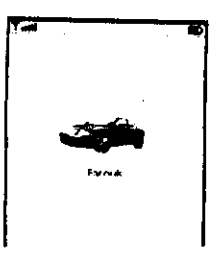


Figure5-4: SplashScreen

5-1-3.MainMenu

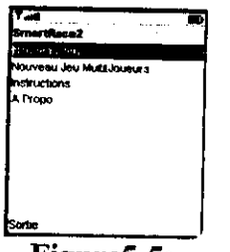


Figure5-5: MainMenu

Le menu de Notre jeu offre quatre choix plus une commande qui sert a quitter le jeu (en bas à gauche), on peut basculer entre les choix en utilisant les flèches haut et bas pour monter ou descendre dans le menu, après navigation dans le menu on doit sélectionner un choix alors en va détailler chaque choix dans les sections suivantes.
En cas de création d'un nouveau jeu un choix va s'ajouter automatiquement "Continue" qui va permettre à un joueur de reprendre sa Partie.

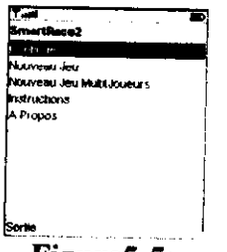


Figure5-5: MainMenu + Continue

5-1-4. AboutScreen

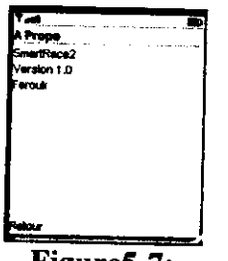


Figure5-7: AboutScreen

C'est le Screen relatif au choix numéro 4 ("A Propos") il a comme but d'afficher le nom du jeu ("SmartRace2"), la version du jeu ("Version 1.0") et en fin le constructeur du jeu ("Farouk"), il est doté d'une commande nommée "Retour" qui sert a revenir dans le MainMenu autrement dit le menu principal.

5-1-5.InstructionScreen

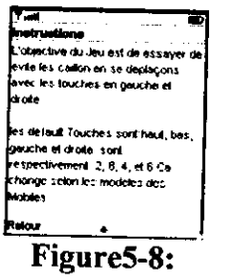


Figure5-8: InstructionScreen

C'est le choix numéro 3, c'est comme un manuel d'utilisation du jeu, il donne une brève description des règles du jeu, les touches de directions et une aide complet relative au jeu. On peu basculer vers le haut et le bas en utilisant les flèches situées sur le clavier du téléphone. Il est doté aussi d'une commande retour qui sert à revenir vers le menu principale.

5-1-6. Game

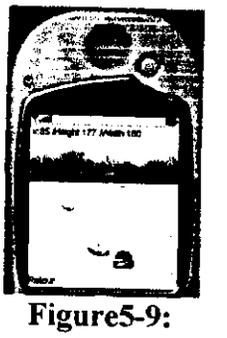


Figure5-9: Game

C'est le choix numéro 1, la on est dans le cœur du jeu avec une voiture, une route et des obstacles, en haut il y a des informations concernant la position **X** , **Y** de la voiture et **Height** , **Width** relative a la taille du Screen(la taille du Canvas) .Quand la voiture percute un obstacle on aperçoit une légère secousse lors de l'affichage du mot "Collision" en haut, et dans le cas ou la voiture touche les limites de la route il y a un texte qui s'affiche ,en haut ,annonçant qu'une règle est déclenchée. Il est aussi muni d'une commande Retour pour revenir au menu principal.

5-1-7. MultiGameMenu

C'est le choix numéro 2, il mène à deux sélections possibles:
- la première est d'être client et de chercher un serveur qui héberge le jeu.
- la deuxième possibilité est d'être serveur ce qui permet d'héberger le jeu.

Ce Screen est dote aussi d'une commande Retour afin de revenir au menu principal.

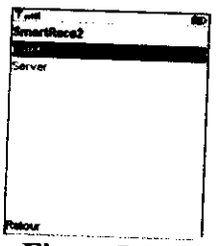


Figure5-10: MultiGameMenu

5-2. Conception & Implémentation sous J2ME

Comme décrit précédemment au par avant notre MIDlet est conçus en 4 parties, nous présenterons le diagramme de classe en premier ensuite on détaille chaque partie du travail.

5-2-1. Le Diagramme de Classe

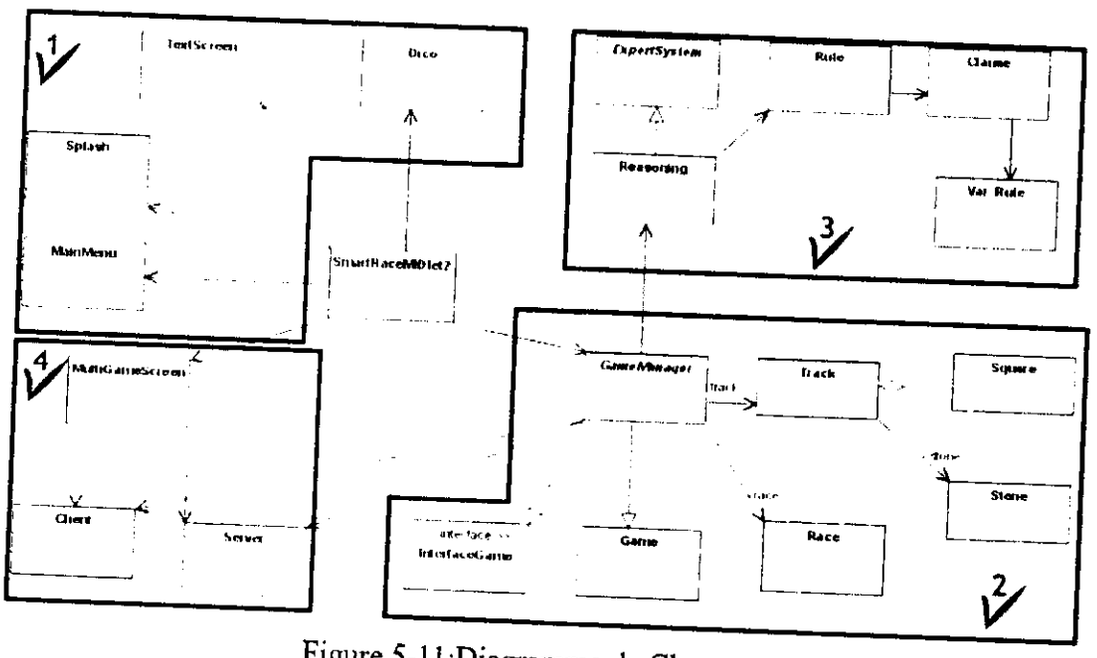


Figure 5-11:Diagramme de Classes

La Classe **SmartRaceMIDlet2** gère l'affichage de l'interface utilisateur (grâce a la méthode **Display.setCurrent**), décide quand et comment afficher un Screen. Et surtout répond aux appels de retour des Screens pour transiter d'une interface à une autre. Elle centralise le contrôle de l'affichage et rend notre diagramme de classe simple sans flèches entrantes et sortantes relatives a l'affichage de chaque Screen.

SmartRaceMIDlet2 crée d'abord un objet **Dico** qui représente un dictionnaire interne de notre jeu. Ce dernier contient toutes les chaînes utilisées par l'interface utilisateur de la MIDlet. (Pour des raisons de simplification, le diagramme au-dessus ne montre pas tous les Screens utilisant le Dictionnaire.)
Son Constructeur admet comme paramètre 2 chaines de caractères :
La première **microedition.locale** qui représente l'emplacement géographique
Et la deuxième **microedition.encoding** qui représente l'encodage du Mobile.
Cette classe peut s'étendre pour supporter les encodages internationaux mais actuellement par défaut ça ne supporte que français (France) et l'anglais US.

SmartRaceMIDlet2 crée un **MainMenu** qui est responsable de l'affichage du menu .Grâce a cette fonction l'utilisateur peut choisir soit de commencer un nouveau jeu, commencer un jeu avec plusieurs joueurs, de voir l'aide relatif au jeu et de consulter des commentaires a propos du jeu (le nom, version et constructeur du jeu) ou bien de sortir carrément du jeu.

SmartRaceMIDlet2 crée aussi un **MultiGameScreen** ce dernier est responsable du menu de création d'un jeu multijoueur relatif au choix numéro 2 du **MainMenu**, il donne le choix a l'utilisateur de ce connecter a un autre terminal en tant que Client, ou bien d'opérer en tant que Serveur .Il permet aussi de retourner au **MainMenu** grâce a la commande Retour.

Avant que **SmartRaceMIDlet2** ne lance le menu elle crée un **SplashScreen** ce dernier s'affichera pendant 4 secondes maximum après cela il cédera sa place automatiquement au **MainMenu**.

La classe **TextScreen** est responsable de l'affichage des 2 Screen **InstructionScreen** Et **AboutScreen**, le **InstructionScreen** affiche les instructions ou bien l'aide relative au jeu tandis que le **AboutScreen** affiche les informations concernant le jeu, comme son nom, sa version et son constructeur.

Quand l'utilisateur choisie un nouveau jeu la **SmartRaceMIDlet2** crée une instance de la class **Game**, ça tache est de contrôler et d'assembler les différentes parties du jeu (c'est le coeur de notre application), les classes **Track** et **Race** sont implémentés par **Game** comme leurs noms l'indique. Elles sont responsables respectivement du dessin de la route et de la voiture.

La classe **Game** crée aussi une **RuleBase** (la base de règles en français) c'est notre system expert en parcourant les règles qui sont implémentées grâce a la class **Rule** elle-même décomposes sous forme de **Clause** c'est le rôle de la classe portant le même nom. La Clause contient une variable relative a la règle c'est la tache de **Var_Rule**.

Et pour la partie réseau il y a deux classes respective au mode de communication **Client** ou bien **Server**, le serveur gère la transmission de donnes tandis que le client est directement liée au serveur de communications, on va reprendre cette partie avec plus de détails dans les sections suivantes.

5-2-2. L'Architecture de SmartRace2

5-2-2-1. La classe SmartRaceMIDlet2

C'est la classe principale de notre jeu elle hérite de la classe **javax.microedition.midlet** (pour plus de détails voir la section 3-5. Les Midlets) elle est responsable de l'affichage des Screens pour l'interface utilisateur, elle centralise les appels d'aller et retour aux différents écrans de haut niveau grâce a la fonction **Display.getDisplay(this).setCurrent(le_Screen_a_afficher)** :
Elle possède quatre variables membres
Le **Dico1** responsable de l'affichage des textes des Screens
Le **mainMenu** affiche le menu du jeu sous forme de liste de choix
Le **multigamescreen** responsable du menu d'un jeu multijoueur
En fin, le **gamemanager** qui représente le jeu en lui même

```
public class SmartRaceMIDlet2
  extends MIDlet{

  private final Dico dico1;
  private final MainMenu mainMenu;
  private final MultiGameScreen multigamescreen;
  private Game gamemanager;
```

Voici le Constructeur qui crée une instance de **Dico** admettant comme paramètre les paramètres locaux du téléphone et l'encodage du texte
Il crée aussi un **MainMenu** avec comme paramètre **this** la MIDlet car elle gère l'interface et un **Dico** pour l'affichage du texte du menu utilisateur

```
  public SmartRaceMIDlet2()
  {
    dico1=new Dico(System.getProperty("microedition.locale"),
                   System.getProperty("microedition.encoding"));

    mainMenu = new MainMenu(this, dico1 );
    multigamescreen = new MultiGameScreen(this,dico1);
  }
```

La méthode **startApp** est appelée directement après le lancement de la MIDlet (voir section 3-5) elle se lance en créant un **splashScreen** en spécifiant le Screen suivant (le menu utilisateur) qui s'affichera automatiquement après 4 secondes

```
  public void startApp()
  {
  Displayable current = Display.getDisplay(this).getCurrent();

    if (current == null)
    {
      //Pour le premier affichage on appelle le Splash.
      // Le main menu s'affichera après 4 secondes.
      Splash splash = new Splash(this, mainMenu);
```

```
            Display.getDisplay(this).setCurrent(splash);
            splash.start(); // va disparaître après un temps limite
        }
        else
        {
            Display.getDisplay(this).setCurrent(current);

        }

    }
```

Méthode appelée En cas de pause de la MIDlet (voir section 3-5) est :

```
    public void pauseApp()
    {

    }
```

Le destructeur de la MIDlet

```
    public void destroyApp(boolean unconditional)
    {
    }
```

Ces deux méthodes sont appelées respectivement, après l'affichage du Splash et TextScreen, pour passer au Screen suivant du MainMenu

```
    void splashDone(Displayable next)
    {
        Display.getDisplay(this).setCurrent(next);
    }

    void textScreenClosed()
    {
        Display.getDisplay(this).setCurrent(mainMenu);
    }
```

Les fonctions suivantes concernent les appels de Retour du **MainMenu,** comme on a expliqué précédemment l'interface utilisateur est centralise au tour de notre MIDlet. La fonction **mainMenuAbout** est responsable de l'affichage du choix "A propos", elle récupère le nom de la MIDlet et sa version et son vendeur, crée une instance de **TextScreen** en passant tout ça comme paramètre. En fin, elle l'affiche grâce a la fonction **Display.getDisplay(this).setCurrent(aboutScreen)**

```
void mainMenuAbout()
  {
      String name = getAppProperty("MIDlet-Name");
    String version = dico1.getString(Dico.LABEL_VERSION) +
              " " + getAppProperty("MIDlet-Version");
    String vendor = getAppProperty("MIDlet-Vendor");

    String about = name + "\n" + version + "\n" + vendor;
    String title = dico1.getString(Dico.LABEL_ABOUT);
    String back = dico1.getString(Dico.LABEL_BACK);

    TextScreen aboutScreen = new TextScreen(this, title, about, back);
    Display.getDisplay(this).setCurrent(aboutScreen);
  }
```

Même chose que la précédente fonction mais cette fois ci on récupère les instructions relatives au jeu à partir de l'objet **Dico**.

```
  void mainMenuInstructions()
  {
      String title = dico1.getString(Dico.LABEL_INSTRUCTIONS);
    String back = dico1.getString(Dico.LABEL_BACK);
    String text = dico1.getString(Dico.TEXT_INSTRUCTIONS);
    //        text += dico1.getString(Dico.TEXT_INSTRUCTIONS_GAUGE);
    TextScreen screen = new TextScreen(this, title, text, back);
    Display.getDisplay(this).setCurrent(screen);
  }
```

La méthode suivante est appelée pour sortir du jeu

```
  void mainMenuExit()
  {
      destroyApp(false);
      notifyDestroyed();
  }
```

Création d'un nouveau jeu simple avec comme paramètre la MIDlet (en cas de pause ou retour) et le Dico pour les textes du jeu. Elle appelle la méthode **start_game()** Pour le lancement du Thread du jeu et l'ajoute le choix "**Continue**" au menu principal.

```
void mainMenuNewGame()
  {
    gamemanager = new GameManager(this,dico1);
    Display.getDisplay(this).setCurrent(gamemanager);
    gamemanager.start_game();
    mainMenu.addContinue();
  }
```

Le passage du menu principal au menu responsable du jeu multi joueurs

```
void mainMenuMultiGameScreen()
{
   Display.getDisplay(this).setCurrent(multigamescreen);
}
```

C'est pour poursuivre le jeu

```
void mainMenuContinue()
{
   Display.getDisplay(this).setCurrent(gamemanager);
}
```

C'est la méthode de l'appel retour du jeu elle comporte comme paramètre un booléen qui indique si le jeu est **GameOver** ou non (**GameOver** veut dire que le joueur a perdue cette partie), si oui on supprime le choix "Continue" si non on rajoute "Continue" au menu, a la fin le menu principal est affiché.

```
void gameMainMenu(boolean isGameOver)
{
   // si le isGameOver on
   // supprime 'Continue' du MainMenu.
   if (isGameOver)
   {
      mainMenu.deleteContinue();
   }
   else
   {
      // highlight 'Continue' in the main menu
      mainMenu.selectContinue();
   }
   Display.getDisplay(this).setCurrent(mainMenu);
}
```

5-2-2-. Les Composants de L'Interface de haut niveau

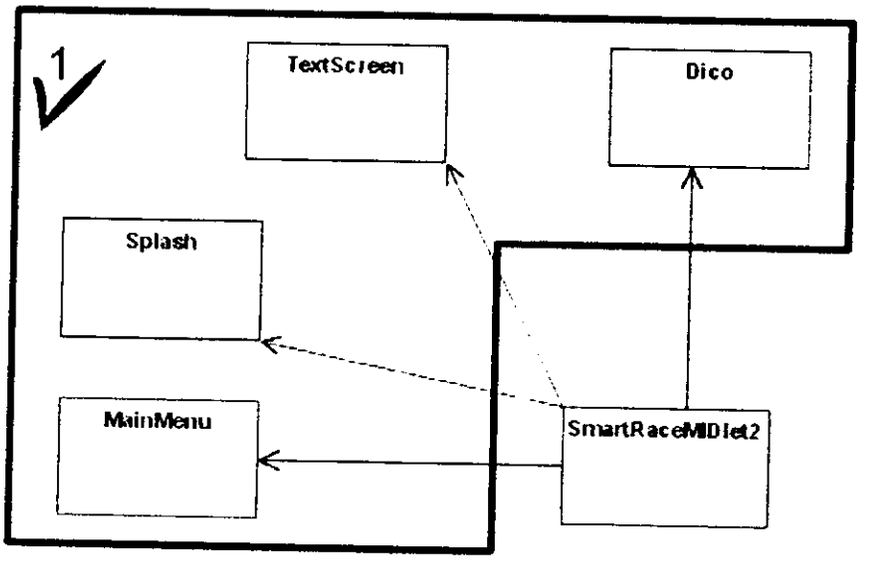


Figure 5-12:Classes Interface Utilisateur de Haut Niveau

a- La classe SplashScreen

Cette classe a pour but d'afficher une image illustrant le jeu et aussi le nom du vendeur de ce dernier, après écoulement de quatre secondes le **SplashScreen** va céder place automatiquement au menu principal. Comme elle doit dessiner une image alors elle hérite de la class **Canvas** (voir section 3-6-3-1), elle possèdent une référence au MIDlet , un **Displayable next** pour l'affichage du Screen suivant (probablement le menu principale), une image et un "timer" pour compter le nombre de secondes.

Figure 5-14:SplashScreen

```
class Splash
   extends Canvas
{
   private final SmartRaceMIDlet2 midlet;
   private final Displayable next;

   private Image image;
   private Timer timer = new Timer();
```

Le constructeur admet comme paramètre une référence sur la MIDlet et le Screen suivant, il lit une image en traitant une exception au préalable.

```
   Splash(SmartRaceMIDlet2 midlet, Displayable next)
   {
      this.midlet = midlet;
      this.next = next;
```

```
        try
        {
          image = Image.createImage("/splash.png");
        }
        catch (IOException e)
        {
          image = null;
        }
    }
```

La méthode **paint** héritée de la classe Canvas (voir section 3-6-3-1),elle dessine l'image et le texte du vendeur lu à partir de notre MIDlet

```
    public void paint(Graphics g)
    {
        ...
      // 1) Arrière plan blanc
      g.setColor(0xFFFFFF);
      g.fillRect(0, 0, canvasWidth, canvasHeight);

      // 2) dessine l'image
        ...
      g.drawImage(image, (canvasWidth / 2), y, anchor);

      // 3) teste si il y a de la place pour dessiner le Texte du vendeur
        ...
      y += (imageHeight + 4); // 4 pixels distance entre image & string
      if (canvasHeight >= (y + font.getHeight()))
      {
        g.setFont(font);
        g.setColor(0x000000); // text NOIR
        String vendor = midlet.getAppProperty("MIDlet-Vendor");
        g.drawString(vendor, (canvasWidth / 2), y, anchor);
      }
    }
```

b- La classe MainMenu

Cette classe permet d'afficher un menu avec une liste de choix, le choix "Continue" s'ajoute automatiquement après création d'un nouveau jeu comme le montre figure suivante:

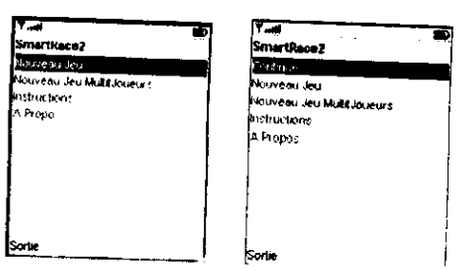


Figure 5-14:MainMenu

Elle hérite de la classe **List** (voir section 3-6-2-2) afin d'afficher la liste des choix et implémente l'interface **CommandListener** (voir section 3-6-2-6) pour la gestion d'événement, elle possède une référence sur la MIDlet, une autre sur le Dico et un boolean pour indiquer si il y a "Continue" ou non.

```
class MainMenu
  extends List
  implements CommandListener
{
  private final SmartRaceMIDlet2 midlet;
  private final Dico dict;
  private boolean hasContinue = false;
```

Le Constructeur possède deux paramètres un sur la MIDlet et le deuxième pour le Dico, il ajoute les choix respectifs au menu et rajoute une commande pour sortir du jeu.

```
  MainMenu(SmartRaceMIDlet2 midlet, Dico dico1 )
  {
    super(midlet.getAppProperty("MIDlet-Name"), List.IMPLICIT);
    this.midlet = midlet;
    this.dict = dico1;

    append(dict.getString(Dico.LABEL_NEWGAME), null);
    append(dict.getString(Dico.LABEL_NEWMULTIGAME), null);
    append(dict.getString(Dico.LABEL_INSTRUCTIONS), null);
    append(dict.getString(Dico.LABEL_ABOUT), null);

    addCommand(new Command(dict.getString(Dico.LABEL_EXIT),
                  Command.EXIT, 1));
    setCommandListener(this);
  }
```

Cette fonction ajoute le choix "Continue" au menu

```
void addContinue()
  {
    // add Continue a la liste du menu , si elle n'est pas présente
    if (!hasContinue)
    {
      insert(0, dict.getString(Dico.LABEL_CONTINUE), null);
      hasContinue = true;
    }
  }
```

Cette fonction supprime le choix "Continue" du menu

```
void deleteContinue()
  {
    // supprimme 'Continue' de la liste du menu ,si elle est pressente
    if (hasContinue)
```

```
        {
            this.delete(0);
            hasContinue = false;
        }
    }
```

Cette méthode traite les événements relatifs au choix du menu (tirer de l'implémentation du **CommandListener** voir section 3-6-2-6)

```
public void commandAction(Command command, Displayable d)
{
    if (command == List.SELECT_COMMAND)
    {
        String selected = getString(getSelectedIndex());
        if (selected.equals(dict.getString(Dico.LABEL_CONTINUE)))
        {
            midlet.mainMenuContinue();
        }
        else if (selected.equals(dict.getString(Dico.LABEL_NEWGAME)))
        {
            midlet.mainMenuNewGame();
        }
        else if (selected.equals(
            dict.getString(Dico.LABEL_NEWMULTIGAME)))
        {
            midlet.mainMenuMultiGameScreen();
        }
        else if (selected.equals(dict.getString(Dico.LABEL_ABOUT)))
        {
            midlet.mainMenuAbout();
        }
        else if (selected.equals(
            dict.getString(Dico.LABEL_INSTRUCTIONS)))
        {
            midlet.mainMenuInstructions();
        }
    }
    else
    {
            midlet.mainMenuExit();
    }
}
```

c- La classe TextScreen:

Ce Screen affiche tout simplement un texte, le Screen est utilisé soit pour l'affichage du texte d'aide du jeu, ou bien pour les information relatives au jeu

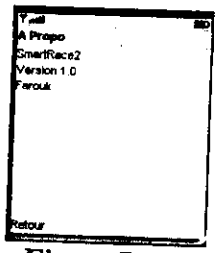


Figure5-15: AboutScreen

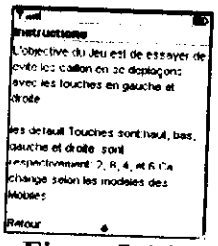


Figure5-16: InstructionScreen

Il possède une référence sur notre MIDlet, il hérite de la classe **Form** (voir section 3-6-2-3) et implémente l'interface **CommandListener** pour la gestion d'événement.

```
class TextScreen
   extends Form
   implements CommandListener
{
   private final SmartRaceMIDlet2 midlet;
```

Le Constructeur admet comme paramètre une référence sur la MIDlet, un texte qui représente le Titre de **la Form**, un Texte que va contenir la **Form** et un texte pour le **label**.

```
   TextScreen(SmartRaceMIDlet2 midlet, String title, String text,
         String closeLabel)
   {
      super(title);
      this.midlet = midlet;
      append(text);
      addCommand(new Command(closeLabel
                  , Command.BACK, 1));
      setCommandListener(this);
   }
```

C'est la section responsable du traitement d'événement, elle fait appelle a la fonction de la MIDlet **midlet.textScreenClosed()** qui réaffiche le menu principale.

```
   public void commandAction(Command c, Displayable d)
   {
      // pour la commande 'close'.
      midlet.textScreenClosed();
   }
```

d- La classe Dico

Cette classe contient tout les textes affichés dans l'interface utilisateur, pour le moment elle supporte l'anglais (US) et le Français (France).Les textes sont tous stockés dans un tableau de chaîne de caractères **strings,** avec plusieurs constantes, chaqu'une d'entres elles correspond a une chaîne dans notre tableau (exp:strings

[LABEL_ABOUT]=le label du bouton about) et ainsi cette classe pourra automatiquement supporter les langages de chaque région.

```
class Dico
{
  private String[] strings;
  private static short ix = 0;
  final static short NUM_IDS = ix;
  private static Dico instance = null;
  final static short LABEL_ABOUT = ix++;
  final static short LABEL_BACK = ix++;
.....
```

Voici un tableau récapitulatif de ses variables avec les valeurs respectives de chaque langue:

Variable	Texte en Anglais (US)	Texte en Français (France)
LABEL_ABOUT	About	A Propos
LABEL_BACK	Back	Retour
LABEL_CONTINUE	Continue	Continue
LABEL_EDIT	Edit	Modifier
LABEL_EXIT	Exit	Sortie
LABEL_INSTRUCTIONS	Instructions	Instructions
LABEL_NEWGAME	Modify	Modifier
LABEL_MODIFY	New game	Nouveau Jeu
LABEL_OFF	off	off
LABEL_ON	on	on
LABEL_NEWMULTIGAME	New Multi Game	Nouveau Jeu Multi Joueurs
LABEL_CLIENT	Client	Client
LABEL_VERSION	Version	Version
LABEL_SERVER	Server	Server
TEXT_GAME_BASE_DESTROYED	Base destroyed!	Base détruite
TEXT_GAME_BLOCKS	Blocks	Obstacles
TEXT_GAME_QUIT_PROMPT	Use soft key to quit	Utiliser le clavier pour quitter
TEXT_GAME_RESUME_PROMPT	Press a key to resume	Utiliser le clavier pour reprendre
TEXT_GAME_LIVES	lives	vies
TEXT_GAME_YOU	You	vous
TEXT_GAME_YOU_WON	You won!	Vous gagnez!
TEXT_GAME_YOU_LOST	You lost	Vous perdez
TEXT_INSTRUCTIONS	The objective of the game	L'objective du Jeu est de .....
TEXT_INSTRUCTIONS_GAUGE	A gauge may sometimes	Par fois un indicateur sera afficher .....

Figure 5-17:Les variables de la classe Dico

Le constructeur admet comme paramètre deux chaines de caractères, la première représente la région (pays), et la deuxième a l'encodage de la langue.
En fonction du l'emplacement géographique , la langue respective de chaque pays est chargée (si elle existe) ou bien l'anglais comme langue universelle.

```
Dico(String locale, String encoding)
  {
        if (locale.equals("French_France"))
     {
       strings = stringsFr();
     }else if (locale.equals("En_US"))
     {
       strings = stringsEnUS();
     }
```

5-2-2-3. Les Composants de L'Interface de bas niveau

Ces classes sont responsables de l'implémentation du jeu lui-même (la voiture, la route, l'image de l'arrière plan et les obstacles) et sont organisées comme suite:

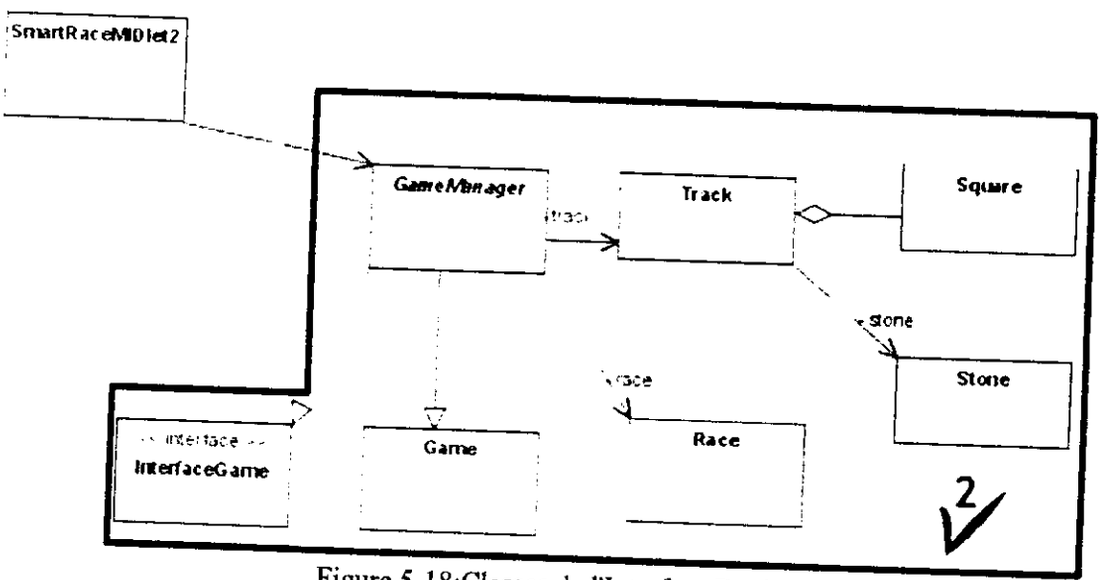


Figure 5-18:Classes de l'Interface Bas Niveau

a- La classe InterfaceGame

Cette interface est conçue pour prévoir l'implémentation d'autre interface comme celle du son et des vibrations de Nokia ou bien siemens, pour le moment elle implémente les interfaces Runnable et CommandListener :

```
interface InterfaceGame
extends Runnable,CommandListener
{

}
```

b- La classe Game

Cette classe abstraite a pour rôle (comme l'interface précédente) d'utiliser d'autres Canvas (voir section 3-6-3-1) comme la GameCanvas supporté a partir de la version MIDP 2.0 ou bien la FullCanvas pour les terminaux Nokia, elle définit aussi le comportement d'un jeu grâce a deux fonctions abstraites **start_game()** et **stop_game().**

Elle peut être hérite par des classes afin d'aider a définir le comportement d'autre jeux.

```
abstract class Game
extends Canvas
implements InterfaceGame
{
abstract void start_game();
abstract void stop_game();
}
```

c- La classe GameManager

C'est la Classe centrale de notre jeu, elle hérite le comportement d'un jeu grâce a la classe Game et elle possède plusieurs variables membres que nous allons détailler ;

Les variables entières **race_x** et **race_y** représentent la position de la voiture tandis que **race_num_image** représente le numéro de l'image à affiché (on vas reprendre cette variable dans la section consacré a la classe **Race**)

Les variables Booléennes **droite** et **gauche** représentent les états des boutons droit et gauche et la variable **tourne** indique que le jeu est en train de s'exécuter.

Le **Thread** est responsable du dessin et de sa synchronisation.
Le **font** représente le font de texte par défaut du jeu.
Une référence sur notre **MIDlet** et **Dico** respectivement responsable du retour et les textes affichés dans le jeu.
Une Classe **Track** responsable du dessin de la route (repris en bas)
Une Classe **Race** responsable du dessin de la voiture (repris en bas)
Un **System Expert** qui seras détaillé dans les sections à venir.

```
public class GameManager
        extends Game
{
private int race_num_image , race_x , race_y;
private boolean tourne , droite , gauche;
private Thread thread;
private Font font=Font.getFont(Font.FACE_MONOSPACE,
                               Font.STYLE_PLAIN,Font.SIZE_LARGE);
```

```
private final SmartRaceMIDlet2 midlet;
private final Dico dict;
private final Track track;
private final Race race;
private final ExpertSystem rb;
```

Son Constructeur admet comme paramètre notre MIDlet pour gérer le Retour au menu et objet de la classe Dico pour l'affichage des Textes.
Initialement les deux boutons droite et gauche sont relâchés et la variable tourne est a vrai, on initialise les cordonnées de la voiture et aussi l'image de voiture de départ, on crée un nouveau objet Reasoning, et après, on initialise les règles du jeu par rapport à notre objet Reasoning. Enfin, on ajoute une commande pour retourner au menu principale.

```
public GameManager( SmartRaceMIDlet2 midlet, Dico dico1 )
{
        tourne=true;
        droite=false;
        gauche=false;
        race_num_image=3;
        race_x=100;
        race_y=getHeight()-40;

        this.midlet = midlet;
        this.dict = dico1;
        this.track = new Track( getHeight() , getWidth() );
        this.race = new Race();
        this.rb=new Reasoning("smart_race2");
        initRuleGame(rb);
        addCommand(new Command(dict.getString(Dico.LABEL_BACK) ,
                        Command.BACK, 1));
      setCommandListener(this);
}
```

Avec ces méthodes on gèrent les événements relatifs aux touches du Canvas (voir section 3-6-3-3) et en met a jour les 2 variables gauche et droite.

```
protected void keyPressed(int keyCode)
    {
      switch(getGameAction(keyCode))
      {
     case 2: // \002
        gauche=true;
        break;

     case 5: // '\005'
       droite=true;
       break;
```

```
        }
    }
protected void keyReleased(int keyCode)
    {
        switch(getGameAction(keyCode))
        {
            case 2: // \002
            gauche=false;
            break;
        case 5: // '\005'
            droite=false;
            break;
        }
    }
```

Et Ici l'événement relatif au retour au menu principal (Voir section 3-6-2-6) est gérée:

```
public void commandAction(Command c, Displayable d)
  {
    // pour la commande 'close'.
    midlet.gameMainMenu(false);
  }
```

d- La classe Race

Cette classe est responsable du dessin de la voiture, il y a sept états différents du positionnement d'une voiture comme la figure suivante l'indique:

Figure 5-19:Les Etats de la Voiture

Ces états sont chargés dans un tableau d'images (voir section 3-6-3-2)

Remarque:

La taille des images est identique, elle désigne par les 2 variables membres **height** et **width**. Le tableau des entiers **pp_image[]** représente les propriéteés d'une image comme le montre la figure suivante:

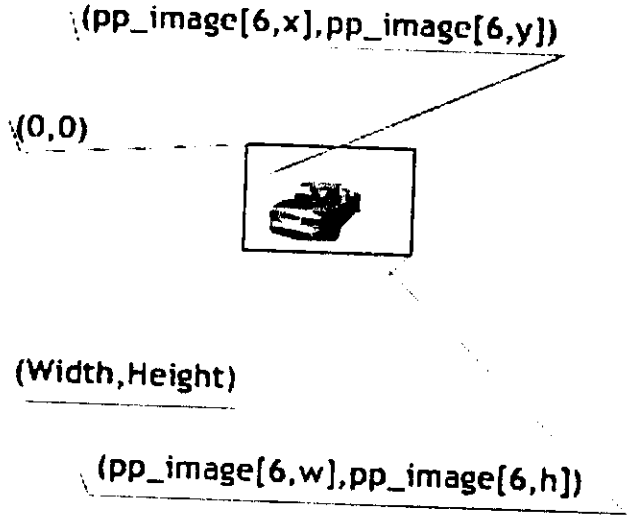


Figure 5-20:Les Proprietes d'une Image

La zone délimitée en noir est la frontière de l'image, tandis que la zone en vert est représente la zone réelle de l'image de la voiture, le point (0,0) représente le coin supérieur gauche de l'image, la taille de l'image est (Width, Height) pour le coin inférieur droit. Le (pp_image[6,x],pp_image[6,y]) représente le point supérieur gauche de l'image du voiture réel tandis que sa taille est représentée par le point (pp_image[6,w],pp_image[6,h])

Remarque: Le numéro 6 passé au tableau pp_image représente le numéro de l'image dans le tableau d'image de la class Race.

Voici la définition de la classe Race:

```
public class Race{

private Image image1[];
public final int height;
public final int width;
public final int w=0,h=1,x=2,y=3;
//*****************Tableau des Ppt des images*********************//
public int pp_image[][]={
//--image 0-----//
{35,23,8,12},
//--image 1-----//
{31,23,10,12},
//--image 2-----//
{26,23,14,12},
//--image 3-----//
{25,23,17,12},
//--image 4-----//
{26,23,19,12},
//--image 5-----//
{28,23,19,12},
```

```
//--image 6----//
{33,23,17,12}
};
```

Son Constructeur lit les images à partir du système de fichier (voir section 3-6-3-2) et retourne une Exception en cas d'erreur.

```
public Race(){
try{
                    image1=new Image[7];

                    for(int i = 0; i<7; i++)
            {
            image1[i] = Image.createImage("/" + (i) + ".png");
            }

           }
         catch(Exception e)
     {
         e.printStackTrace();
     }
 finally{
        height=image1[0].getHeight();
        width=image1[0].getWidth();
        }
}
```

La fonction update dessine la voiture dans un objet graphique passé en paramètre, la variable **state** représente l'état de la voiture et les variables x et y représente ses coordonnées.

```
boolean update(Graphics graphics,int x,int y,int state){

try{
        graphics.drawImage(image1[state], x , y , 0);
}
catch(Exception exception){

        exception.printStackTrace();
        return false;
}
return true;
}
```

e- La classe Track

La classe Track est responsable du dessin de la route, notre route est représentée sous forme de **Square** (carreaux) comme le montre la figure suivante:

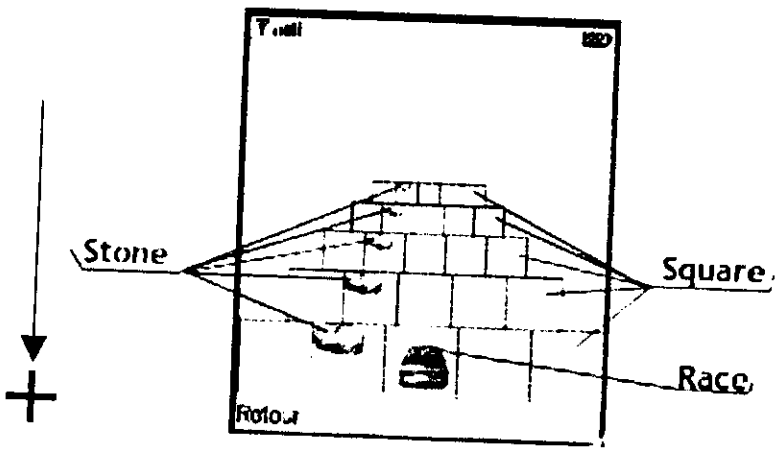


Figure 5-21:La Conception de la Route

Cette classe possède deux entiers qui représentent la taille du Canvas, un objet Stone pour le dessin des obstacles dans la route, la variable **Position** qui nous indique la position courante dans notre circuit , un tableau de square comme la figure précédente indique , une images d'arrière plan et une variable **data** qui va décider des emplacement des obstacles dans notre route .

```
public class Track{

private final int height;
private final int width;
private final Stone stone;
private int position;
private final square tab_square[][];//la table des carreaux
private Image mountain;
private final String data="00003010502040303030303030300000000000000000" ;
```

Notre constructeur admet comme paramètre la taille du Canvas (Voir section 3-6-3-1), il initialise la position courante a Zéro,crée un nouveau objet Stone,charge l'image d'arrière plan en memoir et crée un tableau de square en les organisant comme sur la figure précédente indique.

```
public Track(int height,int width)
{
    ....
}
```

Cette fonction dessine la route c'est à dire l'image d'arrière plan plus les obstacles,en fonction de notre position on lit 5 caractères de la variables **data** voila un exemple qui illustre cela:

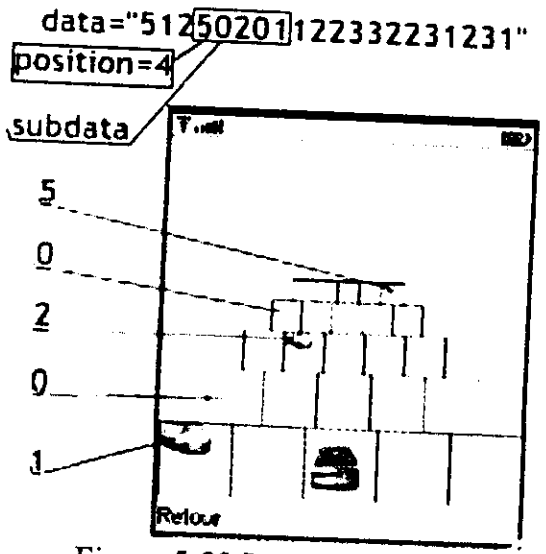


Figure 5-22:Les Obstacles et la Route

Exemple:
Position=4
data="51250201122332231231"
Sub_data="50201"

La variable position correspond a la position dans data, comme on a 5 niveaux de square en prend **sub_data** avec 5 caractères, chaque caractère correspond au rang dans une ligne par exemple le 1 implique qu'on doit dessiner un obstacle au premier square a droite et idem pour chaque caractères de sub_data, voici le prototype de cette fonction:

```
public boolean update(Graphics graphics)
{
        .....
}
```

f- La classe Square

Notre Track est décomposés en 5x5 Square (carreaux), la première ligne (la plus en bas) divisé en 5 et chaque niveau représente 75% du niveau qui le précède. Cela afin de créer un effet de déplacement dans une route .

Voici la classe avec ses deux variables membres **x, y** qui représente la position du point supérieur gauche :

```
class square{
        private int x,y;//pour la position dans le canvas


        public square(int x,int y){
                this.x = x;
                this.y = y;
                }

        public int get_x(){
                return x;
                }

        public int get_y(){
                return y;
                }

        public void set_xy(int x,int y){
                this.x = x;
                this.y = y;
                }

    }
```

g- La classe Stone

Cette classe représente les obstacles sous forme de cailloux (Stone en anglais) ,voici une figure qui les présente .

Figure 5-23:Les Etats d'un Obstacle

La classe Stone contient un tableau d'images des cailloux de chaque niveau.

```
public class Stone{
public Image image1[];
```

Son constructeur charge les cinq images d'obstacles dans notre tableau d'image et gère l'exception en cas d'erreur d'entrée/sortie.

```
public Stone(){
try{
        image1=new Image[5];

        for(int i = 1; i<6; i++)
          {
          image1[i-1] = Image.createImage("/stone" + i +  ".png");
          }

       }
catch(Exception e)
  {
  e.printStackTrace();
  }
```

h- La Méthode Paint() dans GameManager

Elle hérite de la classe Canvas (voir section 3-6-3-1) par le GameManager, elle est appelée avec un délai régulier nommé "**Délai du jeu**" comme l'illustre la figure suivante :

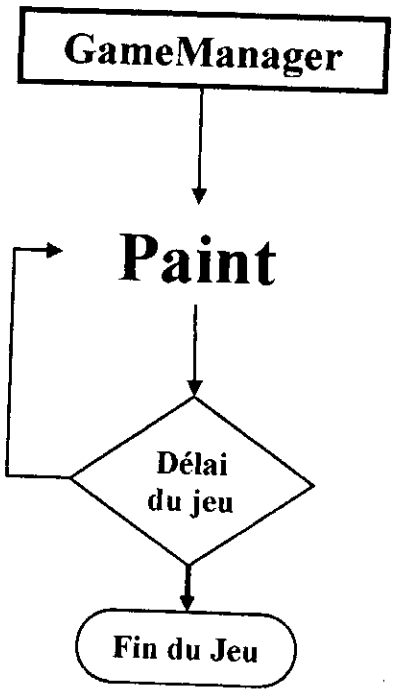


Figure 5-24:Cycle du Jeu

-La surface du dessin est présente en paramètre avec un Graphics (voir section 3-6-3-2) .

public void paint(Graphics graphics)

Les deux paramètres droite et gauche sont testés afin de déterminer les boutons enfoncés, pour déterminer quels états de voiture afficher

if(droite&&race_num_image!=0) race_num_image--;
if(gauche&&race_num_image!=6) race_num_image++;

Si aucun des deux boutons n'est enfoncer l'état de la Voiture Sera Numéro 3 (celle du milieu).

if(droite==false&&gauche==false) race_num_image=3;

Ses deux instructions décalent de cinq pixels lorsque respectivement le bouton droit et gauche serons enfoncé.

if(gauche) race_x-=5;
if(droite) race_x+=5;

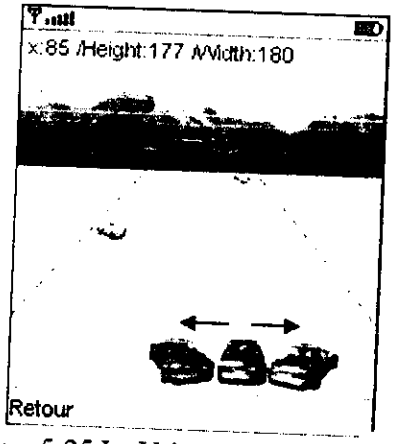


Figure 5-25:La Voiture et les Touches

En suite, le dessin de la route ce fait grâce à l'instruction suivante :

track.update(graphics);

La détection de collision et application des règles du système expert seront détailler dans les sections a venir.

i- La Méthode Run de GameManager

Cette méthode est responsable de la synchronisation du délai de jeu, pour elle tant que la variable booléenne "tourne" est vrai a chaque 1/10 de seconde on rappelle la méthode **Paint** qui va redessiner le Screen, et traiter une exception en cas d'erreur.

```
public void run()
{
while(tourne){
        try
        {
         Thread.sleep(100L);
         synchronized(this)
                {
                 repaint();
                 serviceRepaints();
                }
        }
       catch(Exception exception)
            {
              exception.printStackTrace();
            }
             }
}
```

j- Les Méthodes start_game() et stop_game:

Ces méthodes sont héritées du comportement de la classe abstraite **Game**, la première comme son nom l'indique démarre le jeu en créant et lancent un **Thread**

```
void start_game(){

      if(thread == null)
        {
         thread = new Thread(this);
         thread.start();
        }
      }
```

La deuxième fait le travail inverse, elle termine le jeu en détruisant la référence relative au **Thread.**

```
void stop_game(){

     if(thread != null)
            {
             thread = null;
            }

}
```

k- La Détection des collisions

C'est fait avec la méthode **is_collision()** de la classe **GameManager**
Il y a deux manières de détecter une collision:

1-Detecter une collision par rapport au limites des images : Ça ci est léger par rapport au processeur mais désagréable en matière de vision, car le profile MIDP (voir section 3-3) admet des images en format PNG (Portable Network Graphics) avec des niveaux de transparence quand il y a collision entre les parties transparente non visible pour l'extérieur le jeu détecte comme même une collision

2-Detecter une collision par rapport des pixels réels: Oui c'est efficace par rapport à extérieur mais c'est trop lourd en matière d'exécution, car en utilise des terminaux mobiles avec des capacités limites en matières de traitement.

- La Solution adaptes:
On a crée notre surface virtuelle qui délimite le vrai objet, c'est dessine en vert dans la figure 5-26 comme ça on a trouve un compromis entre utilisation du processeur et l'interface pour les joueurs.

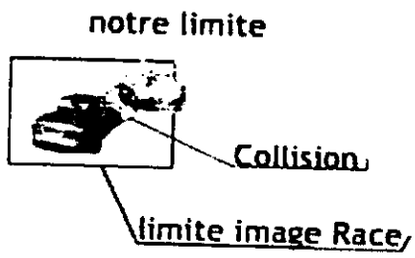


Figure 5-26:Collision

Voici l'implémentation de cette méthode
Elle teste les cordonnes des images et retourne **true** si il y a collision et **false** dans le cas contraire:

```
private boolean is_collision(int x1,int w1,int x2,int w2)
  {
   if(x1==-1)return false;
   if((x1<=x2)&&(x2<=x1+w1)) return true;
   if((x1>x2)&&(x1<=x2+w2)) return true;
   return false;
  }
```

- Détection de Collision dans la méthode Paint de GameManager:

Dans la méthode Paint on appelle la fonction is_collision avec en paramètre les cordonnées du "stone" s'il y en a un ,(-1,-1) sinon, et aussi les coordonnées réel de la voiture .Si il a collision on dessine le texte collision au Screen et en rajoutent l'effet de collision au voiture.

```
public void paint(Graphics graphics)
{
......
//---------COLLISION-----------------//
if(is_collision(        track.now_stone.get_x(),     Stone.width,
        race_x+race.pp_image[race_num_image][race.x],
        race.pp_image[race_num_image][race.w]
                                                ))){
                        graphics.drawString("Collision",40,20,0);
                        race_y-=2;
                        }else race_y=getHeight()-40;
                //-------------------------------------//
..........
}
```

I- Réglage du déroulement de la route et l'affichage de la voiture:

On savons que le parcours dans la route se fait par rapport à la variable **Position** dans la classe **Track**,pour cela, on a crée une autre variable nommée **real_position** qui va contenir **real_position=position/3** ,c'est cette nouvelle variable qui va décider l'avancement par rapport notre route, et comme ça, le dessin de la voiture se fera trois fois plus que la route, comme montre la figure suivante:

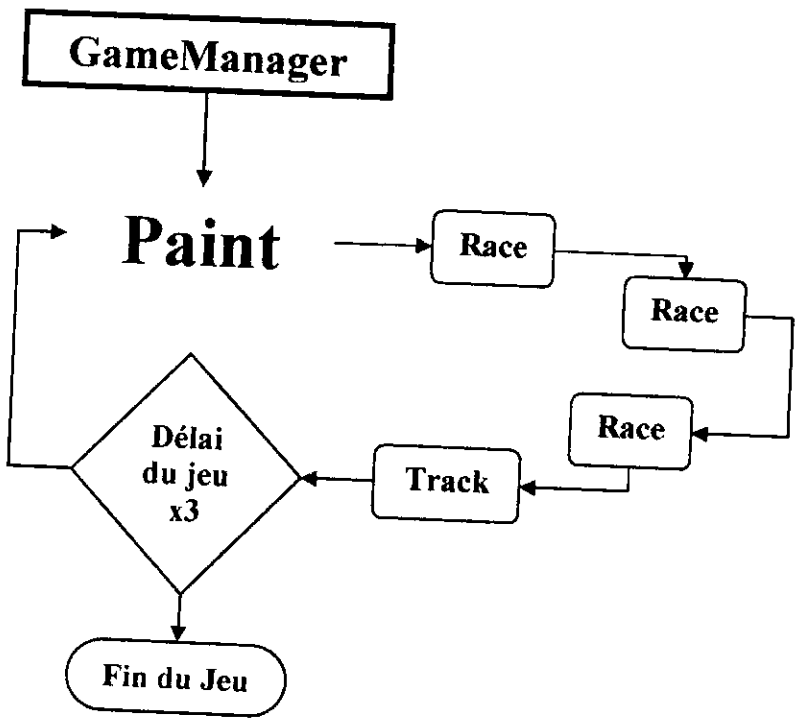


Figure 5-27:Cycle du Jeu x 3

5-2-2-3. Les composants de notre Système Expert

a- La classe Clause

Les clauses sont employées dans les parties antécédentes et conséquentes d'une règle. Une clause se compose habituellement d'un **Var_Rule** du côté gauche, d'une **condition**, qui examine l'égalité, supérieur, inférieur, et différent. Par exemple la règle:

track_limit_droit: **IF** race_x>"200" **THEN** race_x= "200"

Contient deux clauses. La première clause antécédente se compose **du Var_Rule** "race_x", **la condition** ">", et un String "200". L'autre clause est idem.
Une clause contient également un vecteur **des règles** qui contiennent cette **clause**, un booléen **conséquent** qui indique si la clause apparaît dans l'antécédent ou le conséquent de la règle, et un entier **truth** qui indique si la clause est vraie, faux, ou inconnue ('1=true' or '0=false' or '-1=null').

```
public class Clause{

static int number=0; //pour compter les clauses
private String name;  //le nom de la clause
private int ID;       //identificateur de la clause
private Vector rules;  //les règles utilisant cette clause
private Var_Rule var_rule;//la variable Rule
private char condition;  //la condition:'='or'<'or'>'....
private String value_rule;//la valeur cad le partie droite de la clause
private int truth;//la valeur de vérité de la clause:'1==true'or'0==false'or'-
                  // 1==null'
private boolean consequent ; // si la règle est dans les clauses conséquentes
```

Le constructeur de clause prend un string pour son nom une **Var_Rule** pour le côté à gauche, une **condition**, et un string **value_rule** pour le côté droit.

```
public Clause(String name,String var_rule,char condition,String value_rule){
        rules=new Vector() ;
        this.var_rule=new Var_Rule(var_rule,"");
        this.name=name;
        this.condition=condition;
        this.value_rule=value_rule;
        truth=-1;
        }
```

La classe **de clause** contient quatre méthodes. La méthode *d'addRuleRef()* est employée par le constructeur **de règle** pour enregistrer **la règle** avec cette **clause**.

```
void AddRule(Rule Rl){rules.addElement(Rl);}
```

La méthode **check()** teste la clause. Si elle est employée comme clause conséquente, nous renvoyons une valeur nul.
Si la variable du côté à gauche est nul, nous renvoyons également la valeur nulle, parce que ça valeur de vérité ne peut pas être déterminée.
Si la variable est non nulle, nous testerons cette variable pour renvoyer la valeur de vérité résultante.

```
public int Check(){

        if (consequent) return-1 ;

        if (var_rule == null) {

                        return truth = -1 ;// la variable est null
          } else {
     switch(condition)
     {
      case '=': truth = (var_rule.get_value()==value_rule)?1:0 ; break ;
      case '!': truth = (var_rule.get_value()!=value_rule)?1:0 ; break ;
      case '<': truth = inf(var_rule.get_value(),value_rule) ; break ;
      case '>': truth = sup(var_rule.get_value(),value_rule) ; break ;
        }
    return truth;
    }
      }
```

b- La classe Var_Rule

Cette classe représente une variable dans une règle, elle possède un String pour son nom et un autre string pour sa valeur.
Son constructeur est simple il initialise ses deux variables membres, elle a aussi des méthodes pour la lecture et écriture de ces variables par l'extérieur.

```
public class Var_Rule{
private String name;  //le nom de la variable
private String value;
//-------------------------------------//
public Var_Rule(String name,String value){
        this.name=name;
        this.value=value;
}
public void set_name(String name) { this.name=name;}
//-------------------------------------//
public String get_name(){return name;}
//-------------------------------------//
//-------------------------------------//
public void set_value(String value) {this.value=value;}
//-------------------------------------//
public String get_value(){return value;}
}
```

c- La classe Rule

La classe Rule est défini pour représente une seule règle, elle contient une référence sur le **ExpertSystem** auquel elle appartient, son nom, sa valeur de vérité et un booléen indique si elle a été inférée ou non , un tableau des clauses antécédentes et une seule clause conséquente .

```
public class Rule{
ExpertSystem rb ;//reference sur la ExpertSystem sur la quelle elle appartient
String name ;//le nom de la règle
int truth;//la valeur de vérité de la clause:'1==true'or'0==false'or'-1==null'
boolean fired=false;//indique si la règle a étais inférer ou non
Vector antecedents ; // Tableau des Clause antécédents
Clause consequent ;    //la Clause conséquente
```

Son constructeur admet en paramètre son ExpertSystem, son nom, un vecteur contenant ses clauses antécédentes et une clause conséquente.

```
public Rule(ExpertSystem Rb, String Name, Vector lhs, Clause rhs) {
  rb = Rb ;
  name = Name ;
  antecedents = lhs ;
  consequent = rhs ;
  rhs.AddRule(this) ;
  rb.clauseList.addElement(rhs);
  rhs.set_consequent(true) ;
  Enumeration enu = lhs.elements();
  while(enu.hasMoreElements()){
        Clause tmp=(Clause)enu.nextElement() ;
        rb.clauseList.addElement(tmp);
  }
  rb.ruleList.addElement(this) ;
  truth = -1 ;
}
```

Sa méthode **check()** teste ces clauses antécédentes .si l'une d'elles est a null ça veux dire ayant la valeur (-1) , elle retourne null . Si l'une d'elles est a la valeur false (0) ,elle retourne false et si toutes les clauses vaut true (1) elle retourne true .

```
public int check() {
  for (int i=0 ; i < antecedents.size() ; i++ ) {
  if (((Clause)(antecedents.elementAt(i))).get_truth() == -1) return -1 ;
  if (((Clause)(antecedents.elementAt(i))).get_truth() == 1 ) {
     continue ;
  } else {
     return truth = 0 ;
  }
  }
  return truth = 1;
}
```

La méthode **fire()** indique si la règle a été inférée et vrai ,elle met a jour les deux variables **truth** et **fire** a vrai et modifie la clause conséquente a vrai

```
void fire() {
 truth = 1 ;
 fired = true ;
 consequent.set_truth(1);
}
```

La fonction **reset()** réinitialise la règle a une autre utilisation, en mettons **truth** , **fired** et la clause conséquente a false (-1) ,ainsi que toutes ses clauses antécédentes

```
void reset() {
 truth = -1 ;
 fired = false ;
 consequent.set_truth(-1);
 for (int i=0 ; i < antecedents.size() ; i++ )
 ((Clause)(antecedents.elementAt(i))).set_truth(-1);
}
```

d- La classe ExpertSystem

Cette classe abstraite définit le comportement d'un système expert a base de règles, elle possède 2 vecteurs le premier pour la liste des clauses et le deuxième pour ses règles, elle implémente deux méthodes abstraites la première **forwardChain()** pour le chaînage avant (voir section 2-8-3-1) et la deuxième pour la réinitialisation du système.

```
abstract class ExpertSystem
{
        Vector clauseList;//la liste des Clauses
        Vector ruleList ;//la listes des règles
        abstract void forwardChain();
        abstract void reset();
}
```

e- La classe Reasoning

Cette classe hérite directement du comportement de la classe **ExpertSystem**, elle possède une chaîne qui représente son nom .

```
public class Reasoning extends ExpertSystem{
 String name ;//le nom de la RuleBase
```

Son constructeur alloue deux vecteurs pour la liste des règles et clauses hérite de la classe **ExpertSystem** et admet comme paramètre son nom.

```
public Reasoning(String name) {
        this.name = name;
        ruleList=new Vector();
        clauseList=new Vector();
                                }
```

f- L'implémentation du Chaînage avant

Le chaînage avant (voir section 2-8-3-1) est implémentée comme une fonction **forwardChain()** dans classe **Reasoning**. La méthode crée d'abord le vecteur de **conflictRuleSet et** elle appelle La méthode **match()** avec un paramètre booléen vrai pour forcer un premier essai de toutes les règles dans la base de règle. Elle retourne en suite *le* **conflictRuleSet** initial, un vecteur des règles qui sont déclenchées et pourrait être inféré. Nous metterons alors une boucle *while()*, qui fonctionne jusqu'à ce que nous ayons *un* **conflictRuleSet** vide. À l'intérieur de la boucle, nous appelons d'abord la méthode **selectRule()**, avec *le* **conflictRuleSet** comme paramètre. La méthode **selectRule()** recherche et retourne la meilleur règle a inférer,alors Nous appelons **Rule.fire()** pour mettre a vrai la règle et la clause conséquente, et puis on essais de nouveau toutes les clauses et règles qui se rapportent à la variable mise à jour.
Nous appelons le **match()** encore, mais cette fois on passe une valeur booléenne fausse par paramètre. Ceci indique a la fonction **match()** de regarder seulement les valeurs de vérité de règle,sans les examinées.

```
public void forwardChain() {
  Vector conflictRuleSet = new Vector() ;
  // test1
  conflictRuleSet = match(true); // selection
      //System.out.println("conflictRuleSet.size:"+conflictRuleSet.size());
  while(conflictRuleSet.size() > 0) {

   Rule selected = selectRule(conflictRuleSet);
   selected.fire() ;
   System.out.println("regle appliqué:  "+selected.get_name());

       conflictRuleSet = match(false);
  }
}
```

Maintenant regardons avec plus de détail les différentes méthodes.
Reasoning.match() la méthode prend un paramètre booléen simple. Elle crée un vecteur de règles **matchList** . Si le paramètre **test** est vrai, il appelle **Rule.check()** pour examiner toutes les clauses antécédentes de règle et pour placer la valeur de vérité de la règle.
Si *l'essai* est **test**, *le* ***match()*** regarde simplement la valeur de vérité de la règle courante. Si la règle est vraie, et elle n'a pas été déjà inférée, elle l'ajoutera au vecteur *de* **matchList**. Sinon, nous passerons à la prochaine règle sur **le ruleList**.

```
public Vector match(boolean test) {
  Vector matchList = new Vector() ;
  Enumeration enu = ruleList.elements() ;
  while (enu.hasMoreElements()) {
   Rule testRule = (Rule)enu.nextElement() ;
   if (test) testRule.check() ;
   if (testRule.truth == -1) continue ;
   // fire
   if ( (testRule.truth == 1 ) &&
      (testRule.fired == false) ) matchList.addElement(testRule);
  }
  return matchList ;
}
```

Reasoning.selectRule() la méthode prend un vecteur des règles comme paramètre représentant l'ensemble des règles en conflit. Notre fonction est assez simple, elle recherche la meilleure règle à inférer, pour elle une meilleure règle qui possède le plus grand nombre d'antécédents .

```
public Rule selectRule(Vector ruleSet) {
  Enumeration enu = ruleSet.elements() ;
  long numClauses ;
  Rule nextRule ;
  Rule bestRule = (Rule)enu.nextElement() ;
  long max = bestRule.numAntecedents() ;
  while (enu.hasMoreElements()) {
    nextRule = (Rule)enu.nextElement() ;
    if ((numClauses = nextRule.numAntecedents()) > max) {
      max = numClauses ;
      bestRule = nextRule ;
    }
  }
  return bestRule ;
}
```

5-2-2-4. Les composants de la partie Communication Bluetooth

Notre application utilise **RFCOMM** pour le transfères de données (voir section 4-4), il existe deux autre manières **L2CAP** et **OBEX** (pour plus d'information consulter le help de JSR-082 sur la reference [32], pour résumer la **RFCOMM** est utilisée pour la transmission en Stream , **L2CAP** pour les paquets et **OBEX** pour les objets ou fichier .

a- La classe Client

Cette classe possède un discoveryAgent pour la recherche des terminaux, remoteDevices pour classer les terminaux découverts, uuidSet pour l'authentification avec le Server et serviceUrl qui représente l'url de communication. Elle Implémente aussi l'interface DiscoveryListener (voir section 4-4).

```
class Client implements DiscoveryListener {
  private DiscoveryAgent discoveryAgent;
  private RemoteDevice[] remoteDevices;
  private UUID[] uuidSet;
  private String serviceUrl;
```

Le constructeur de la classe Client recherche les terminaux Bluetooth a proximité et en suite rechercher les services disponibles sur ce terminal pour cela:
En premier, on retourne le **LocalDevice** (un terminal locale) et on obtient un **DiscoveryAgent** (pour la recherche).

```
public Client() {
   try {
LocalDevice localDevice = LocalDevice.getLocalDevice();
discoveryAgent = localDevice.getDiscoveryAgent();
```

Maintenant on démarre la recherche des terminaux Bluetooth a proximité avec la méthode **startInquiry** de la classe **DiscoveryAgent** (voir section 4-4-2-1)

```
discoveryAgent.startInquiry(DiscoveryAgent.GIAC,this); } catch (Exception e) {
    System.out.println(e);
  }
 }
```

DiscorveryAgent.GIAC veut dire "General/Unlimited Inquiry Access Code" pour rechercher tout les terminaux Bluetooth avec pas de exception. En dernier en gèrent l'exception relative à ce traitement.

L'interface DiscoveryListener nous obligent a implémenter quatre méthodes ,chaque une d'entres elles fait pour une tache spécifique:

DeviceDiscovered: elle est déclanchée après la découverte d'un terminal durant une recherche des terminaux Bluetooth, dans son corps on enclenche la découverte des services.

```
public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
   try {
    // les Informations du Device
    System.out.println("Device Discovered");
    System.out.println("Major Device Class: " + cod.getMajorDeviceClass() + "
Minor Device Class: " + cod.getMinorDeviceClass());
    System.out.println("Bluetooth Address: " +
btDevice.getBluetoothAddress());
    System.out.println("Bluetooth Friendly Name: " +
btDevice.getFriendlyName(true));
      // recherche des Services
   uuidSet = new UUID[1];
   uuidSet[0] = SmartRaceMIDlet2.RFCOMM_UUID;
   int searchID = discoveryAgent.searchServices(null,uuidSet,btDevice,this);
  } catch (Exception e) {
```

```
    System.out.println("Device Discovered Error: " + e);
  }
 }
```

inquiryCompleted appeler après avoir terminer de rechercher ou de découvrir les terminaux Bluetooth, Elle affiche le message "InquiryCompleted"

```
public void inquiryCompleted(int discType) {
  System.out.println("InquiryCompleted");
 }
```

servicesDiscovered: appeler après avoir découvert un services durant une recherche de services,elle lit l'url de communication et affiche le message "ServiceDiscovered".

```
public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
  System.out.println("ServicesDiscovered");
  // dans notre application il y a un seul service
  for(int i=0;i<servRecord.length;i++) {
    serviceUrl = servRecord[i].getConnectionURL(0,false);
  }
 }
```

serviceSearch: elle est déclancher après avoir termine la recherche des services dans un terminal découvert,
si la reponseCode égale a:
SERVICE_SEARCH_ERROR qui veut dire une erreur, elle affiche "SERVICE_SEARCH_ERROR"
SERVICE_SEARCH_COMPLETED qui veut dire une recherche avec succès, elle affiche l'url de connection et transmet le message **"Client dit Bonjour le monde"**au serveur, en dernier elle traite une exception éventuellement déclancher pendant la transmission
SERVICE_SEARCH_TERMINATED qui veut dire la recherche des services est terminé,elle affiche "SERVICE_SEARCH_TERMINATED"
SERVICE_SEARCH_NO_RECORDS qui veut dire pas de service dans ce terminal, Elle affiche le message "SERVICE_SEARCH_NO_RECORDS"
SERVICE_SEARCH_DEVICE_NOT_REACHABLE qui veut dire que le terminal n'est pas accessible, Elle affiche le message
"SERVICE_SEARCH_DEVICE_NOT_REACHABLE"

```
public void serviceSearchCompleted(int transID, int responseCode) {
   if(responseCode == SERVICE_SEARCH_ERROR)
      System.out.println("SERVICE_SEARCH_ERROR\n");
   if(responseCode == SERVICE_SEARCH_COMPLETED) {
      System.out.println("SERVICE_SEARCH_COMPLETED\n");
      System.out.println("Service URL: " + serviceUrl);
      StreamConnection conn = null;
      try {
            String msg = "Client dit Bonjour le monde";
            conn = (StreamConnection)Connector.open(serviceUrl);
            OutputStream output = conn.openOutputStream();
```

```
                output.write(msg.length());
                output.write(msg.getBytes());
                output.close();
                System.out.println(SmartRaceMIDlet2.readData(conn));
        } catch (Exception ex) {
                System.out.println(ex);
        } finally {
                try {
                        conn.close();
                } catch (IOException ioe) {
                        System.out.println("Error Closing connection " + ioe);
                }
                }
  }
  if(responseCode == SERVICE_SEARCH_TERMINATED)
    System.out.println("SERVICE_SEARCH_TERMINATED\n");
  if(responseCode == SERVICE_SEARCH_NO_RECORDS)
    System.out.println("SERVICE_SEARCH_NO_RECORDS\n");
  if(responseCode == SERVICE_SEARCH_DEVICE_NOT_REACHABLE)
 System.out.println("SERVICE_SEARCH_DEVICE_NOT_REACHABLE\n");
 }
```

b- La classe Server

Cette classe possède deux variable **input , output ,conn** et **notifier** pour la communication Stream (voir section 4-3) les deux première sont respectivement pour lire et écrire ,les deux autres pour la réalisation de la communication, l'url de la communication et un booléen indiquent si elle est initialisée ou non.

```
public class Server implements Runnable {

 LocalDevice localDevice;
 ServiceRecord serviceRecord;
 InputStream input;
 OutputStream output;
 StreamConnectionNotifier notifier;
 StreamConnection conn;
 private boolean isInit;
 private static String serverUrl;
```

Son Constructeur initialise l'url de communication, crée et lance un nouveau Thread pour attendre les réponses des clients.

```
public Server() {
  isInit = false;
  serverUrl = "btspp://localhost:" + SmartRaceMIDlet2.RFCOMM_UUID +
              ";name=rfcommtest;authorize=true";
  Thread thread = new Thread(this);
  thread.start();
 }
```

Cette fonction teste si le serveur est initialisé ou pas , sinon elle effectue ça en créant un **StreamConnectionNotifier** à l'aide de **Connector** (voir section 4-3) et un LocalDevice. Après il reste en pause jusqu'à ce que une Transmission du client aura lieu, quand c'est la cas le serveur lit le message grâce a la méthode **SmartRaceMIDlet2.readData();** qui sera détaillée dans la section suivante et réponds avec le message "**Server Respond Bonjour**" .en cas d'exception elle est gèrée en fin de code suivant:

```
public void run() {
    if (!isInit) {
    // Initialisation du serveur
    try {
     conn = null;
     localDevice = LocalDevice.getLocalDevice();
     localDevice.setDiscoverable( DiscoveryAgent.GIAC );
     notifier = (StreamConnectionNotifier)Connector.open(serverUrl);
    } catch (BluetoothStateException e) {
     System.err.println( "BluetoothStateException: " + e.getMessage() );
    } catch (IOException e) {
     System.err.println( "IOException: " + e.getMessage() );
    }
    isInit=true;
    System.out.println( "Starting Server" );
   }
   try {
    System.out.println("\n\nServer Running...");
        // Pause du Thread jusqu'à que la Transmission aura lieu
    conn = notifier.acceptAndOpen();
    // lit Data
    String msg = SmartRaceMIDlet2.readData(conn);
    System.out.println("Message Reçus du Client: " + msg);
    // la response du server
    msg = "Server Respond Bonjour";
    output = conn.openOutputStream();
    output.write(msg.length()); // length is 1 byte
    output.write(msg.getBytes());
    output.close();
   } catch (Exception ex) {
    System.err.println("Bluetooth Server Running Error: " + ex);
   }
}
```

c- la méthode SmartRaceMIDlet2.readData ()

Cette méthode est utilisé par les deux classes (Client et Server), elle leurs permet de lire un string message. Elle a comme paramètre une **StreamConnection** (voir section 4-4) au quel elle lit un message dans une variable **data** et le retourne par la suite. Elle gère les exceptions En cas d'erreurs, et en fin elle ferme la connection.

```
public final static String readData(StreamConnection conn) {
  InputStream input = null;
  byte[] data = null;
    try {
   input = conn.openInputStream();
   // Probablement une exception si length<=0
   int length = input.read();
   data= new byte[length];
   length = 0;
   // Assemble la variable data
   while (length != data.length) {
    int ch = input.read(data, length, data.length - length);
    if (ch == -1) {
     throw new IOException("on ne peut pas lire data!!!");
    }
    length += ch;
   }
  } catch (IOException e) {
   System.err.println(e);
  } finally {

   // ferme le input stream
   if (input != null) {
    try {
     input.close();
    } catch (IOException e) {
    }
   }
  }
  return new String(data);
}
```

remarque: à chaque délais de jeu, grâce à cette méthode, les positions respectives de chaque voitures serons transmises des deux côtes ,ainsi sera réaliser la partie multi joueurs .

5-3. Les Testes de Notre Jeu

Les tests permettent de réaliser des contrôles de qualité sur notre jeu. Il s'agit de relever les éventuels défauts de conception et d'implémentation. Pour testé notre application, nous avons utilisé le WTK de Sun qui peut simulé toutes les fonctionnalités d'un appareil mobile (Voir Annexe A).Le WTK 2.1 simule le téléchargement d'une application depuis un serveur *web* par son utilitaire *OTA Provisioning* (*On The Air*) qui peut simulé le comportement de notre jeu sur un vrai serveur web, voici les différentes étapes de progression.

Figure 5-28: l'URL d'installation

Figure 5-19: Téléchargement du fichier .Jad

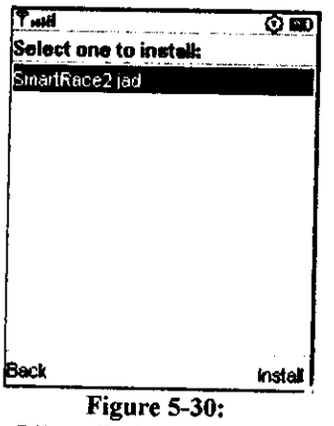


Figure 5-30: L'installation du fichier

En premier, via le WAP on va se connecter a la page web qui contient le fichier .jad (le fichier descriptif de notre application java) , ensuite une liste contenant les fichiers .jad disponible sur le serveur sera affichée.
L'utilisateur pourrait alors choisir l'application à installer.

Figure 5-31: Installation du fichier .jad

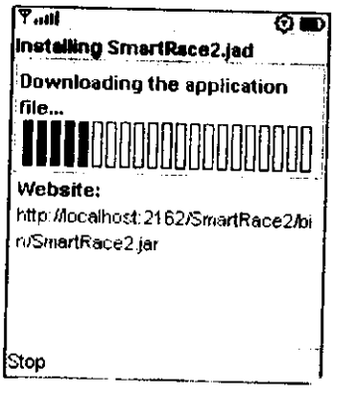


Figure 5-32: Téléchargement du Fichier .Jar

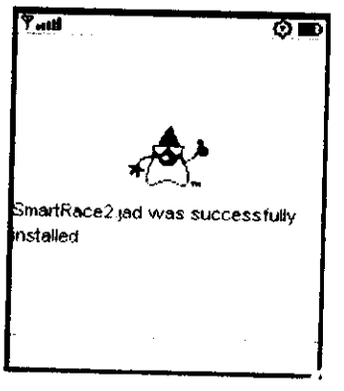


Figure 5-33: Jeu Installer avec Succès

Et maintenant, l'installation se fait en téléchargent un deuxième fichier .jar (java archive) qui représente l'exécutable de notre jeu. Apres être télécharger et installer un écran s'affiche indiquant que la procédure c'est terminé avec sucée.

Après le lancement de notre jeu un menu principal s'affiche,en nous offrant plusieurs choix. On procède avec un nouveau jeu simple (mono joueur). Notre jeu apparaît du premier coup très simple, une voiture ,des obstacles et une route mais dès qu'on percutent un obstacle un message apparais indiquant une collision comme indique la figure suivante :

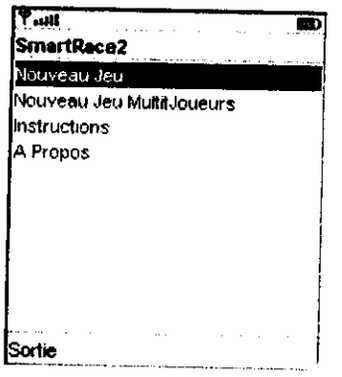


Figure 5-34: Menu Principal

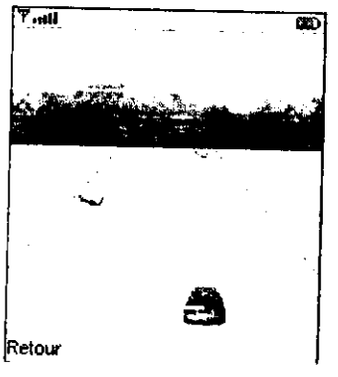


Figure 5-35: le jeu

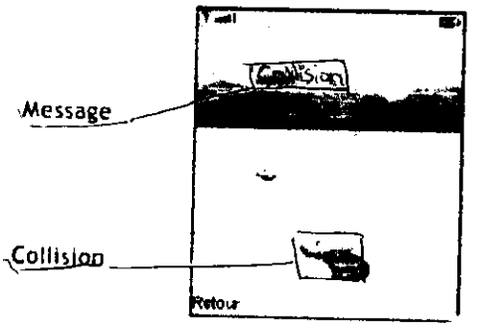


Figure 5-36: Test Collision

Si on essaye de franchir les limites gauche et droite, notre système expert intervient. On déclanchant des règles automatiques comme indiquent les figures suivantes:

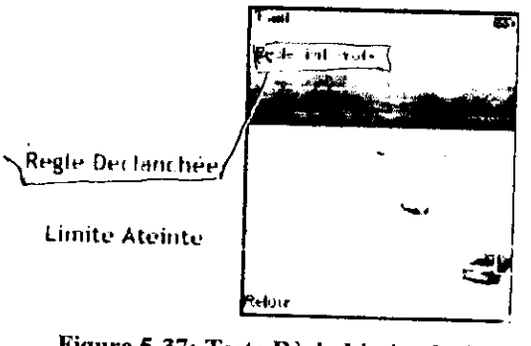


Figure 5-37: Teste Règle Limite droite

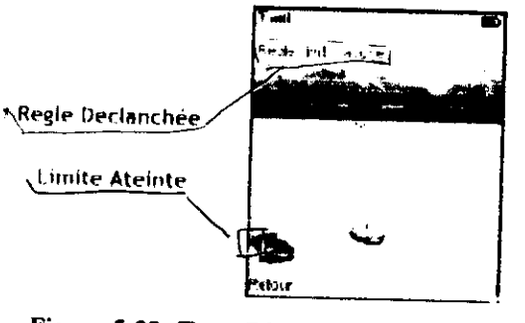


Figure 5-32: Teste Règle Limite gauche

Maintenant, on va teste le jeu multi joueurs pour cela il nous faut deux terminal (notre jeu par défaut supporte 2 joueurs et ça peut être améliorer en futur), un serveur et un client .on débute par la création d'un serveur qui va diffuses ses informations sur un rayon Bluetooth local (10 mètres environ) alors le Screen 1 s'affiche, en répond yes. Ensuite on lance le client (Screen 2) on tapent toujours yes. Après, le Screen 3 apparais en indiquant qu'un client veut se connecter à notre serveur Bluetooth .On va confirmer en répondant yes. Enfin, le jeu démarre automatiquement comme illustre le Screen 4,le schéma suivant décrit tout ça:

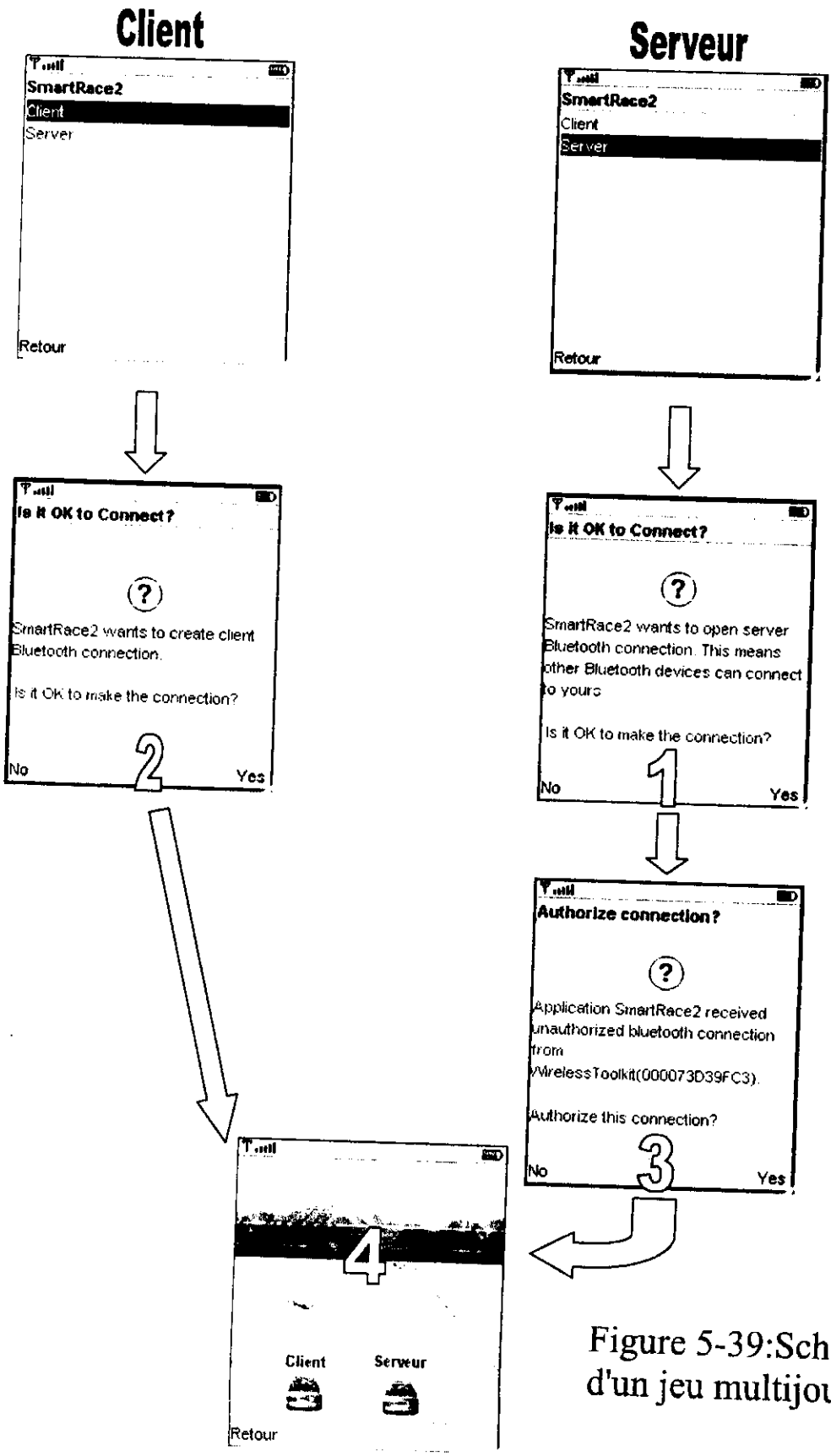


Figure 5-39:Schéma d'un jeu multijoueurs

Conclusion

- Conclusion

Les jeux font partie de la nature humaine, avec l'introduction de l'outil Informatique ils sont devenus la première source de divertissement pour les humains. Mais, les joueurs expérimentes accusent l'ancienne génération de jeux comme **dépassée,** De ce fait, Notre idée est de créer une nouvelle expérience de divertissement, afin d'introduire intelligence dans le domaine des jeux mobiles.

En élaborant ce projet, tout d'abord nous avons réalisés l'interface utilisateur qui permet aux joueurs de naviguer dans les différentes options de jeu. Ensuite, on a implémentés notre jeu avec ses différentes parties: le dessin de la route, son déroulement, la voiture et les obstacles. Après vient l'étape cruciale de notre projet: Le développement du système expert qui introduit l'intelligente de notre travail. Enfin, le travail sera complété en mode multi joueurs par l'implémentation de la technologie Bluetooth dans notre jeu.

Au cours de l'élaboration de notre projet, nous nous sommes heurtés à certaines difficultés, qui tiennent essentiellement aux problèmes suivants:

1. Du aux limitations dans les terminaux mobiles, notre première préoccupation était l'optimisation du code. Cela en réduisant autant que possible la création dynamique des objets, en optimisant le nombre de sprite affichés, en affectant la valeur **null** aux références qui ne sont pas utilisée et en employant autant que possible les variables locales.

2. Devant le manque de l'information. Il a fallu une méthode appropriée pour nous permettre de réaliser notre jeu (SmartRace2).

3. Devant la grande consommation des systèmes experts en matière de ressources, nous avons été contraints d'utiliser des techniques d'optimisations afin de les rendrent moins gourmant en ressources.

4. La rareté des documents expliquant le développement et la méthodologie des jeux multi joueurs utilisant la technologie Bluetooth.

SmartRace2 a été proposé suivant une architecture ouverte facilitant son extension. Parmi les extensions de ce jeu, les perspectives suivantes peuvent être envisagés :

- le développement de l'aspect graphique et sonore du jeu, en rajoutant des effets acoustiques et des animations cognitives.
- Utilisation de la bibliothèque 3D graphique de J2ME (JSR-184).
- Etendre le système expert proposé vers un système multi-agent, les composants de ce dernier, vont interagir grâce à la technologie Bluetooth.
- l'utilisation des patterns dans la modélisation, tel que le MVC (Modèle-Vue-Contrôleur)
- En dehors des jeux, vue que notre système expert est totalement autonome. Il peut être porté vers d'autres applications mobiles tel que les traducteurs.
- Utilisation d'autre technologie de communications sans fil, tel que le Wi-Fi .

Nous avons réalisé une créativité, d'une part évolutive et de l'autre part stratégique. Ayant les possibilités de base qui serviront d'avancer vers des nouvelles visions dans le monde de création des jeux sur mobile et le rendre plus promoteur.

En conclusion, malgré toutes les critiques sur les **systèmes experts (SE),** dans notre étude nous avons montré qu'un SE est un outil capable avec quelques règles de reproduire les mécanismes intelligentes d'un être humain, Associer aux jeux il peut les rendre plus cognitifs et plus réels.

Bibliographie

1. BIBLIOGRAPHIE

[01] **http://perso.wanadoo.fr/jm.doudoux/**
Jean-Michel DOUDOUX
Ingénieur Dans le groupe I.B.M. à partir du 21 Août 1995 (d'abord au sein de la filiale C.G.I. Informatique puis dans la division I.B.M. Global Services à partir de janvier 1999), puis au sein de la SSII Mediaware de juin 2002 à décembre 2004, actuellement salarié au sein de la société Atos Origin depuis janvier 2005.

[02] **J2ME Application java pour terminaux mobiles**
Bruno Delb
EYROLLE 2002

[03] **http://developers.sun.com/techtopics/mobility/apis/articles/bluetoothintro/**
C. Enrique Ortiz, Décembre 2004

[04] **http://www.etis.ensea.fr**
Le site du docteur Arnaud Revel Docteur en Traitement des Images et du Signal Et Maître de conférences à l'ENSEA (Ecole National Supérieur de l'Electronique et ses Applications).

[05] **Micro Java™ Game Development**
David Fox, Roman Verhosek
Addison Wesley April 18, 2002

[06] **Core Techniques and Algorithms in GameProgramming 1st edition**
Daniel Sanchez-Crespo Dalmau, Daniel Sanchez-Crespo
New Riders Games September 8, 2003

[07] **Programmer avec la plate-forme J2ME**
Rogger Riggs,Antero Taivalsaari et Mark VandenBrink
CampusPress 2001

[08] **J2ME game Programming**
MARTIN J. WELLS
Premier Press 2004

[09] **J2ME & Gaming**
Jason Lam
June 30, 2004

[10] **Wireless Java: Developing with Java 2, Micro Edititon**
Jonathan Knudsen
Apress 2001

[11] **MIDP Style Guide for the Java™ 2 Platform, Micro Edition**
Cynthia Bloch, Annette Wagner
Addison Wesley June 10, 2003

[12] **Programming Wireless Devices with the Java™ 2 Platform, Micro Edition, 2nd**
Roger Riggs, Antero Taivalsaari, Jim Van Peursem, Jyri Huopaniemi, Mark Patel, Aleksi Uotila, Jim Holliday Editor
Addison Wesley June 13, 2003

[13] **Enterprise J2ME: Developing Mobile Java Applications**
Michael Juntao Yuan
Prentice Hall PTR October 23, 2003

[14] **Rules of Play: Game Design Fundamentals**
Katie Salen and Eric
The MIT Press © 2004

[15] **Constructing Intelligent Agent Using Java**
Joseph P.Bigus Jennifer Bigus
Wiley 2001

[16] **The Art of Computer Game Design**
Chris Crawford
Mcgraw-Hill Osborne Media January 1984

[17] **Bluetooth for Java**
Bruce Hopkins and Ranjith Antony
Apress © 2003

[18] **developers.sun.com/techtopics/mobility/midp/articles/bluetooth2/index.html**
The Java APIs for Bluetooth Wireless Technology
Qusay H. Mahmoud April 2003

[19] **Developpement d'une Application BlueTooth. Application sur PDA**
SAHBI SAHLOUL
Mémoire de Master dans Université de Nice Sophia Antipolis
www.mips.unice.fr

[20] **www.grappa.univ-lille3.fr**
Cours d'intelligence artificielle et Systèmes experts
François Denis de GRAPPA : Groupe de Recherche sur l'Apprentissage Automatique à l'université de Lille 3

[21] **www.wikipedia.org**
Wikipédia est une encyclopédie multilingue, libre et gratuite. Elle est basée sur un serveur Web utilisant la technologie Wiki. Elle fait partie des 20 sites les plus visités au monde

[22] **www.benhui.net**
C'est un portail de développement - j2me - Bluetooth - mobile 3d – midp1.0 et 2.0

[23] **www.bluetooth.com**
Le site officiel de la norme de communication Bluetooth développez par Nokia, Ericsson, IBM, Intel, Microsoft, Motorola et Toshiba.

[24] **www.forum.nokia.com**
Le site officiel des développeurs sur les terminaux Nokia.

[25] **www.j2meolympus.com**
Un site du développement java sur mobile

[26] **www.microjava.com/**
Un site avec plusieurs articles sur la plate forme j2me

[27] **www.palowireless.com/java/**
Un site avec une section consacrer au développement j2me

[28] **www.j2meforums.com**
Un forum on line sur le développement j2me

[29] **Core J2ME™ Technology & MIDP**
John W. Muchow
Prentice Hall PTR Décembre 21, 2001

[30] **www.sun.com**
Le site officiel du créateur de java.

[31] **www.orangepartner.com**
Le site officiel du créateur de java.

[32] **http://jcp.org/aboutJava/communityprocess/final/jsr082/**
le manuel de java en ligne.

Annexes

A. J2ME Wireless Toolkit

La version 2.0 permet d'utiliser MIDP 1.0 ou 2.0 ainsi que les API optionnels Mobile Media et Wireless Messaging . Il peut être intégré dans d'autres IDE tel que Sun Studio Mobile Edition ou JBuilder.

La version 2.1 permet d'utiliser CLDC 1.1 et l'API J2ME Web service et de développer des applications pour des périphériques qui respectent les spécifications JTWI.

A-1. Installation du J2ME Wireless Toolkit

La version 2.1 du J2ME Wireless Toolkit nécessite la présence sur le système d'un J2SE 1.4 minimum.

Elle permet le développement d'applications répondant aux spécifications de la JSR-185 (Java Technology for the Wireless Industry) qin inclue : CLDC 1.0 + CLDC 1.1, MIDP 1.0 + MIDP 2.0, WMA 1.1 MMAPI 1.1.

Elle permet aussi l'utilisation de la JSR-172 (J2ME Web Services Specification).

Lancer l'application j2me_wireless_toolkit-2_1-windows.exe. Un assistant guide l'utilisateur dans les différentes étapes de l'installation :

- sur la page d'accueil, cliquez sur le bouton « Next »
- sur la page « License Agreement » : lire la licence et si vous l'acceptez, cliquez sur le bouton « Yes »
- sur la page « Java Virtual Machine Location » : le programme détecte automatiquement la présence d'un JDK 1.4 ou supérieure, cliquez sur le bouton « Next »
- sur la page « Choose Destination Location » : sélectionnez le répertoire d'installation de l'application et cliquez sur le bouton « Next »
- sur la page « Select Program Folder » : saisissez ou sélectionner le dossier du menu démarrer qui va contenir les raccourcis vers l'application si celui par défaut ne convient pas. Cliquez sur le bouton « Next »
- sur la page « Start Copying files » : un résumé des options d'installation est affiché. Cliquez sur le bouton « Next »
- les fichiers de l'application sont copiés. Une fois celle ci terminée, la page « Installshield Wizard Complete » s'affiche. Cliquez sur le bouton « Finish ».

L'installation créé les répertoires suivants :

appdb\	contient les bases de données de type RMS des applications
apps\	contient les applications développées comme applications de démo
bin\	contient les outils du WTK
docs\	contient la documentation du WTK et des API
lib\	contient les bibliothèques des API

Figure A-1:La liste des Répertoires

A-2. Premiers Pas

L'outil Ktoolbar est un petit IDE qui permet de compiler, pré-vérifier, packager et exécuter des applications utilisant le profile MIDP. Il ne permet pas l'édition du code des applications : il faut utiliser un éditeur externe pour réaliser cette tâche.

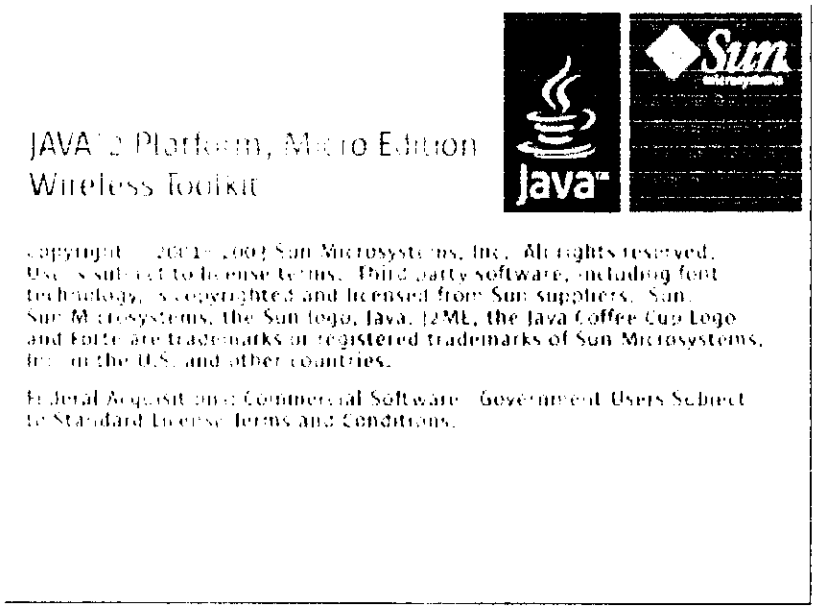


Figure A-2:SpalshScreen de WTK

La première chose à faire pour créer une application est de créer un nouveau projet. Pour cela, il faut sélectionner l'option « File/New Project » du menu ou cliquer sur le bouton « New Project » dans la barre d'outils.

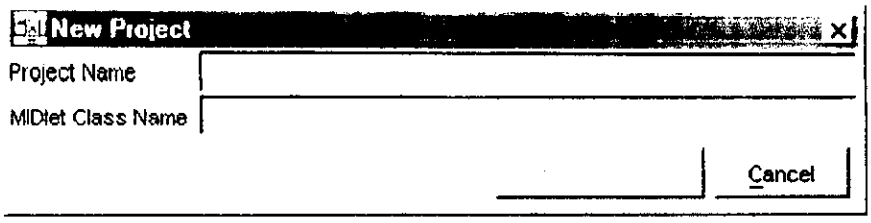


Figure A-3:Nouveau Projet

Il faut saisir le nom du projet et le nom de la classe de la Midlet.

La création du projet permet la création d'une structure de répertoires dans le sous répertoire apps du répertoire du WTK. Dans ce répertoire apps, un répertoire est créé nommé du nom du projet. Ce répertoire contient lui même plusieurs sous répertoires :

%WTK%/apps/nom_projet/bin	contient le fichier jar, jad et le fichier manifest
%WTK%/apps/nom_projet/classes	contient les classes compilées
%WTK%/apps/nom_projet/lib	contient les bibliothèques utiles à l'application
%WTK%/apps/nom_projet/res	contient les ressources utiles à l'application
%WTK%/apps/nom_projet/src	contient les sources des classes

Figure A-4:Les sous Repertoires de Mon projet

La page des propriétés du projet est différente de celle proposée dans la version 1.0. Pour l'utiliser, il faut utiliser l'option « Project/settings » ou cliquer sur le bouton « Settings » de la barre d'outils.

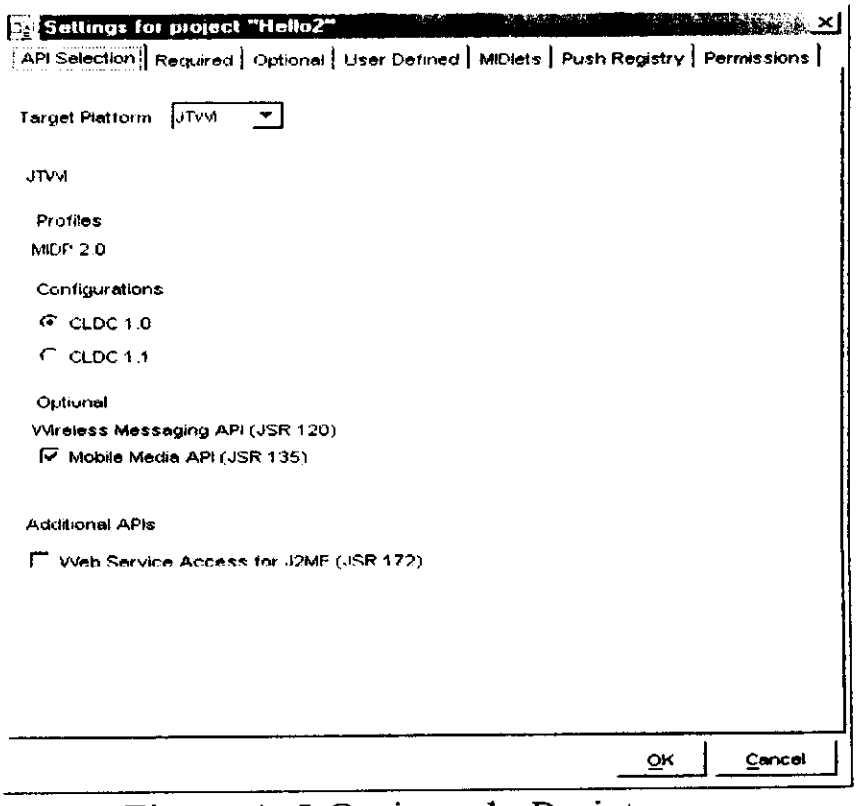


Figure A-5:Options du Projet

L'onglet « API sélection » permet de sélectionner la plateforme cible ainsi que les API particulières qui vont être utilisées par l'application.

Le « target plateform » permet de sélectionner le type de plate-forme cible utilisée :

- JTWI : plate-forme répondant aux spécifications de la JSR-185
- MIDP 1.0 : plate-forme composée de CLDL 1.0 et MIDP 1.0
- Custom : plate-forme personnalisée pour laquelle il faut préciser toutes les API utilisées

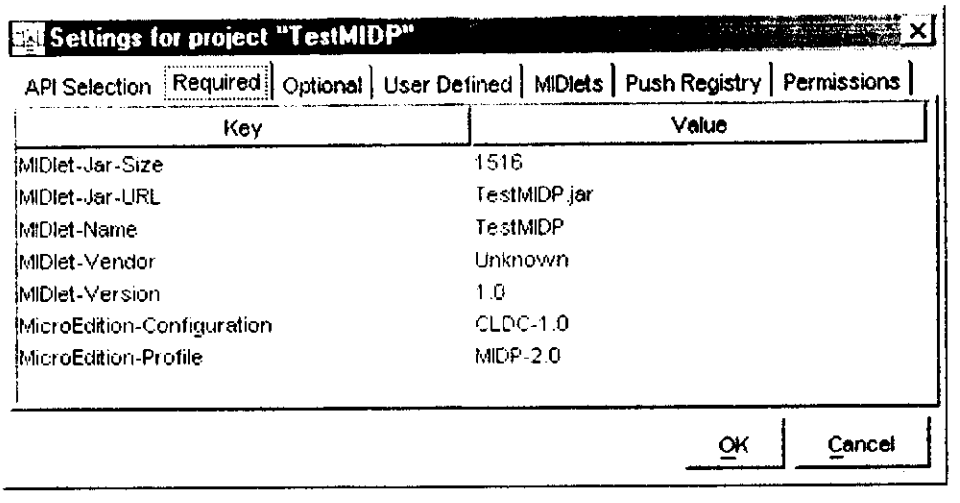


Figure A-6:Détails du MIDlet

L'onglet « Required », « Optionnal » et « User defined » permet de préciser les attributs respectivement obligatoires, optionnels et particuliers à l'application dans le fichier manifest sous la forme de paire clé/valeur.

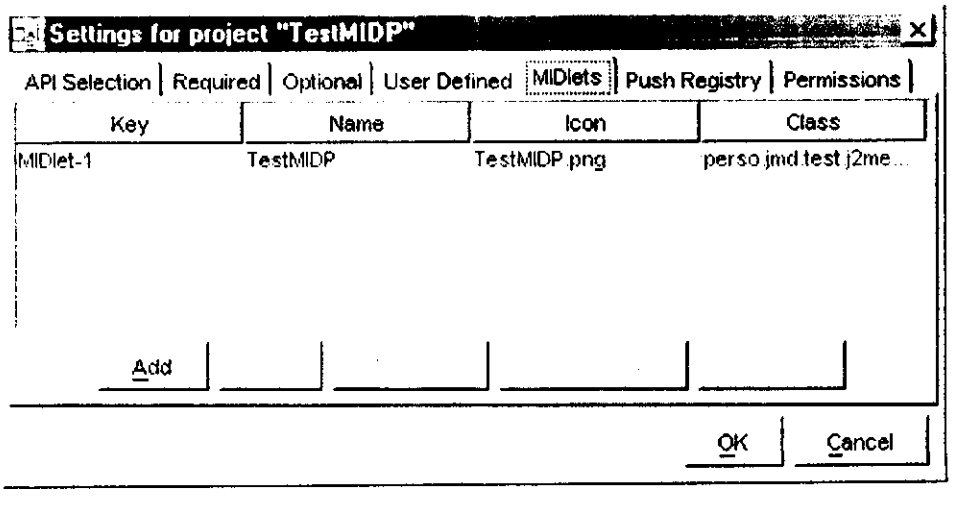


Figure A-7:Attribut du MIDlet

L'onglet « Midlets » permet de saisir les Midlets qui composent la suite de Midlets de l'application.

Pour créer et éditer le code des classes qui composent l'application, il faut utiliser un outil externe dans le répertoire %WTK%/apps/nom_projet/src, en respectant la structure des répertoires correspondant aux packages des classes.

La compilation et la pré vérification des sources se fait en utilisant l'option « Build » du menu « Project » ou en cliquant sur le bouton « Build » .

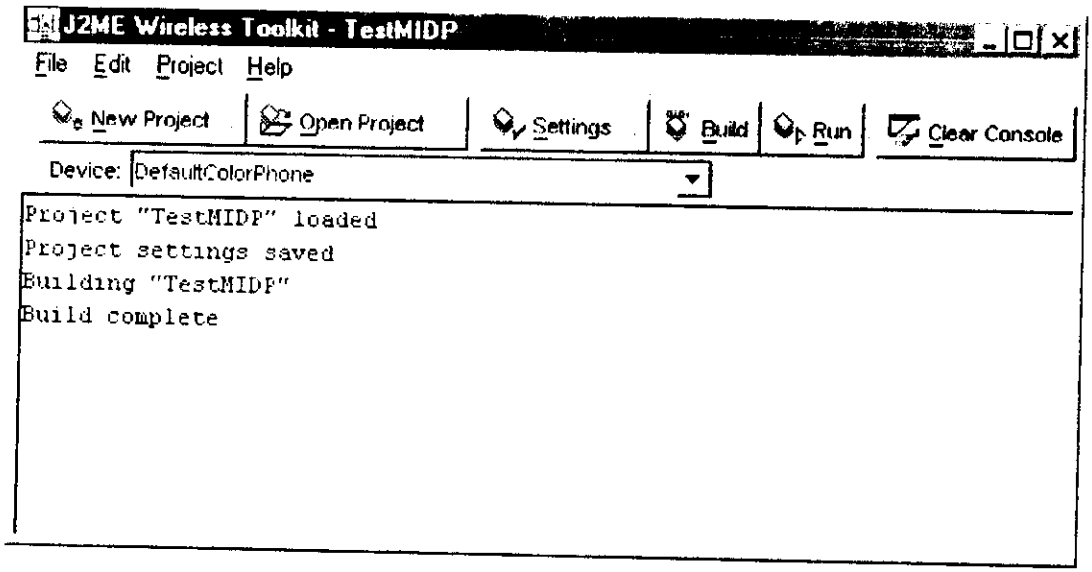


Figure A-8:Compilation du Projet

Si aucune erreur de compilation n'est détectée, il est possible d'exécuter le code en utilisant l'option « Run » du menu « Project » ou en cliquant sur le bouton « Run » de la barre d'outils.

Avant de lancer l'exécution, il est possible de sélectionner l'émulateur de périphérique (device) utilisé pour exécuter le code. Le J2ME Wireless Toolkit 2.1 est fourni avec quatre émulateurs :

- DefaultColorPhone : un téléphone mobile avec un écran couleur. C'est l'émulateur par défaut.
- DefaultGrayPhone : un téléphone mobile avec un écran monochrome
- MediaControlSkin : un téléphone mobile avec des capacités multimédia accrues (video et audio)
- QwertyDevice : un périphérique avec un clavier Qwerty

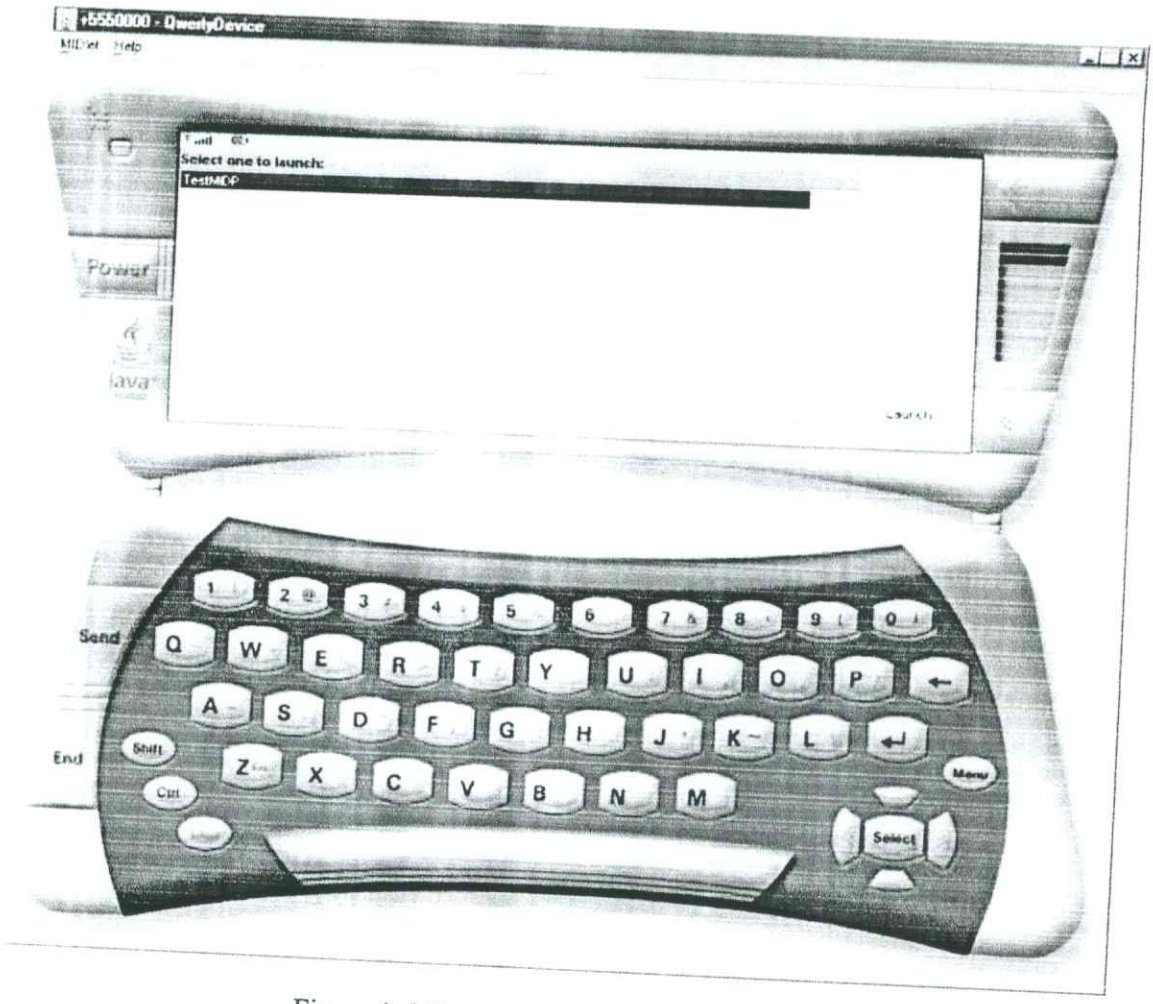


Figure A-9:Execution du Projet

L'option « Clean » du menu « Project » permet de faire du ménage dans les fichiers temporaires générés lors des différents traitements.

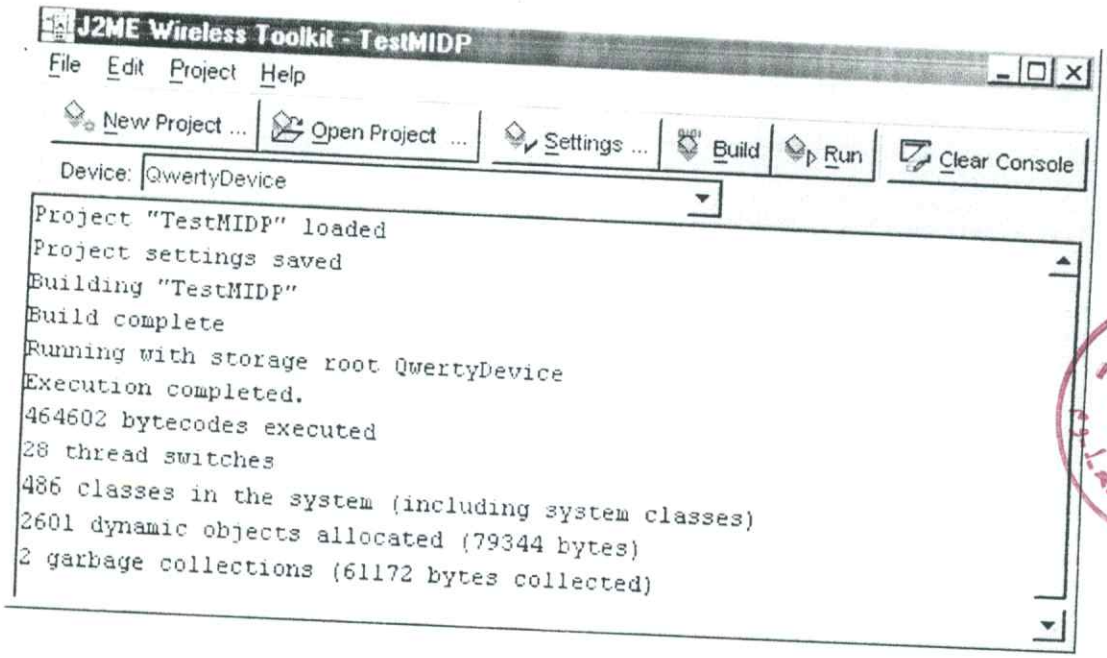


Figure A-10:Après Execution du Projet