Projet conduit à l'université de Bordeaux 1

Bot in Second Life

Individu autonome pour la recherche d'informations sur Second Life

Besse Julien
<julien.besse@etu.u-bordeaux1.fr>

Blondeau Olivier
<olivier.blondeau@etu.u-bordeaux1.fr>

Delahaye Thomas
<thomas.delahaye@etu.u-bordeaux1.fr>

Glacet Christian
<christian.glacet@etu.u-bordeaux1.fr>

Client : Philipe Narbel
Chargé de TD : Pascal Desbarat

Date de première mise en ligne : 12 janvier 2009
Date de dernière mise à jour : 2 février 2009
Version de l'application concernée : V0
Etat : Terminé
Reference en ligne : https://services.emi.u-bordeaux1.fr/projet/viewvc/botinsl2009/trunk/Rapport/rapport-28term-29.pdf?view=log

Résumé

Ce projet consiste en la création d'un bot (entité virtuelle autonome) qui adoptera un comportement humain afin de rechercher des informations dans le Monde de Second Life. Pour se faire, le bot devra se déplacer, discuter et faire des choix de manière autonome. L'utilisateur final de ce projet devra juste donner des caractéristiques personnelles au bot, pour le rendre plus humain, ainsi que l'objet de sa recherche. Le bot sera alors lancé dans ce monde virtuel pour commencer son investigation et trouver une réponse à sa recherche. Le bot fournira un rapport sur sa recherche et sera aussi capable d'améliorer sa base de connaissances grâce à ses différentes recherches. Ce projet sera réalisé à l'aide du projet open-source libSL qui nous a fourni les méthodes pour faire interagir un bot avec le monde de Second Life.

Table des matières

Chapitre 1

Domaine d'application

1.1 Objectifs du projet de programmation

Le but (scolaire) de ce projet est de suivre une démarche qualité assez simple, cela dans le but de nous former pour une meilleure insertion en entreprise. Nous devrons donc tout au long de ce projet éditer des documents liés à la conduite de projet (analyse de besoins, journal de bord, ...). Le but principal de ce projet est donc de produire des documents clairs et précis sur l'avancement du projet, la démarche adoptée, les problèmes rencontrés et les solutions apportées. L'intérêt de la chose étant notamment d'obtenir un projet qui quelque soit son état d'avancement puisse être repris par une autre équipe de développement sans perdre trop de temps. Ou encore de permettre à une personne extérieure d'intégrer l'équipe facilement.

1.2 La recherche d'informations

La recherche d'informations est la science de trouver des informations dans un document de tous type. L'important pour une recherche d'informations est de cibler la demande de l'utilisateur afin de mieux trouver le document. Pour illustrer ce point, quand un utilisateur effectue une recherche via le site Google, il a intérêt à cibler ses mots clés pour trouver une réponse précise.

Pour se faire, il existe plusieurs méthodes dans la recherche d'informations :

1.2.1 Le modèle de l'information Retrieval

Ce modèle consiste à placer un **agent intermédiaire** entre l'usager et la base de documents. Cet agent est chargé de chercher la bonne source d'informations par rapport aux critères de l'usager. Ce modèle s'applique à notre exemple précédent : Google. Le site fait le lien entre l'usager (à travers ses mots clés) et la base de données des sites référencés par Google.

1.2.2 Les représentations basées sur l'exploration

Cette représentation permet à un usager d'effectuer des recherches quand son besoin de départ est assez flou. Il commence avec des documents très généraux et en farfouillant, affine sa recherche pour arriver à des documents plus précis. Pour mettre en parallèle avec Internet, l'usager commence sur un site général sur le domaine puis ce site va l'entraîner sur d'autres liens. Après quelques visites de sites, l'usager va comprendre le sujet, affiner ses besoins et choisir les mots clés qui lui permettront de faire éventuellement une recherche de modèle Retrieval.

1.2.3 La construction d'un espace de recherche

Pour construire un système de recherche d'informations de ce type, l'utilisateur doit définir une représentation de départ, la représentation finale et les actions possibles entre ces étapes. Ce système est assez ciblé au final. Il permet par exemple de chercher comment résoudre le problème de la tour d'Hanoï.

Voici quelques exemples de systèmes de recherche. Ces systèmes basent leur recherche sur un document, c'est à dire dans n'importe quel support où se trouve de l'information. On peut considérer que le monde virtuel de Second Life est un document à part entière. En lisant les principes du bot, vous comprendrez facilement que notre bot sera considéré comme un agent intermédiaire à la recherche faisant le pont entre l'utilisateur qui recherche une information et le monde de Second life, là où se trouvent les réponses (modèle Retrieval). Le modèle se basant sur l'exploration pourra être aussi utile à notre recherche d'informations, notre bot fouinera un peu partout sur le monde de Second Life dans le but d'améliorer sa base de connaissance sur un sujet.

1.3 Second Life (SL)

Second life est un univers virtuel en 3D,qui a débuté en 2003. Cet univers est accessible par Internet mais Second life n'est pas un jeu massivement multijoueur classique. En effet, ici pas de quêtes ou de buts prédéfinis, chaque joueur est libre de parcourir le monde comme bon lui semble. Certains y chercheront un côté social en profitant des nombreuses possibilités de discutions entre joueurs, d'autres privilégierons une démarche économique. D'où le titre du jeu, "Seconde Vie", chaque joueur se crée une identité propre (**avatar**) qu'elle soit proche de la réalité ou inventée de toute pièce.Ensuite, chaque joueur pourra apporter sa pierre à cet univers. Contrairement à beaucoup de jeu en ligne, Second Life donne la possibilité aux joueurs d'apporter des modifications à cet univers, en donnant la possibilité d'acheter des parcelles, y construire des édifices et fabriquer des objets.

Second Life est accessible gratuitement ce qui a permis d'attirer des millions d'avatars à travers le monde. 14 millions de comptes ont été crées depuis 2003, même si en réalité 200 000 à 300 000 comptes seraient joués régulièrement et environ 50 000 joueurs quotidien. Cet engouement populaire et le côté commercial libre a entraîné un véritable déchaînement économique. Un joueur peut acquérir de la monnaie virtuelle (Linden Dollar) contre de vrais dollars, mais il existe beaucoup d'autres moyens pour gagner de l'argent. A l'image de notre société,

des parcelles de terrain qui selon leur côte peuvent être vendu plus ou moins cher. Plus il y a d'avatars passant sur une parcelle , plus la valeur de celle-ci augmentera. Beaucoup de compagnies internationales(Adidas, Dior, Toyota...) ont donc investis dans cet univers pour en faire une vitrine virtuelle. Les joueurs peuvent créer des objets qu'ils peuvent vendre. Les objets sont brevetés, ce qui garantit au joueur de pouvoir profiter sereinement de son commerce. L'univers est donc architecturé et décoré par les joueurs. Ce monde connaît du succès car il permet aux joueurs de donner libre cours à leur imagination que ce soit en termes d'objectifs personnels ou de créativité.

1.4 Linden Lab

La société qui a créé Second Life fut fondée en 1999 par Phillip Rosedate. Jusqu'en Mars , il occupait la place de CEO (Chief Execute Officer) de cette société. A partir de cette date, il a voulu se concentrer plus particulièrement sur l'aspect stratégique et sur l'orientation de l'entreprise. Débarrassé de cette charge administrative, il est néanmoins "chaïman of the board" (Président du Conseil d'Administration). Au début de l'année 2007, Linden Lab décide de passer une partie du code de Second Life Viewer sous la désignation Open Source.

Nous ferons donc un bref résumé de ce qui a pu pousser Linden Lab à changer sa philosophie sur le sujet, ainsi que les risques que cela implique.

Presque quatre ans après son lancement, le mouvement Second Life continue de prendre de l'ampleur. Le client du monde parallèle de Linden Lab vient en effet de passer sous licence GPL, ce qui rend cette partie entièrement recopiable et réutilisable par toute personne souhaitant créer son propre client en gardant comme base le coeur de Second Life ; Le code est disponible sur le site officiel de Second Life : [1]. D'après un des sites officiels [2], nous pourrons comprendre quels sont les intêrets pour Second Life de laisser du code "ouvert" : L'intérêt principal de cette nouveauté est de permettre aux développeurs d'accéder à la version complète de Second Life en y ajoutant du contenu personnalisé tels que des programmes ou des scripts. En laissant ce code Open Source, les créateurs du jeu souhaitaient avant tout rapprocher la communauté de joueurs et les programmeurs de Second Life . Ceci permettra aux développeurs d'obtenir un point de vue précis venant des utilisateurs envers les modifications apportées tandis que ces derniers seront mieux informés des nouveautés et sauront donc comment améliorer leur (le) monde virtuel. Le second intérêt sera d'engendrer une plus forte créativité de la part des joueurs. Le fait qu'une partie du code soit passé en Open Source favorise amplement la création d'outils ou de programmes permettant de créer divers objets sur Second Life . La communauté de joueur pourra alors créer plus facilement des objets au fur et à mesure que le nombre d'outils grandira ! Les concepteurs de ce monde virtuel pensent même à passer une plus grosse partie de leur code d'application "ouvert" s'ils considèrent la première expérience comme réussie, permettant ainsi aux joueurs d'avoir une meilleure connaissance de ce monde. Ceci entraînerait, normalement, une crois-

1. SL `www.secondlife.com`
2. Second Life faq : `http://secondlifegrid.net/technology-programs/virtual-world-open-source/faq`

sance dans le domaine de la création d'outils de toutes sortes pour Second Life . Enfin, pour terminer, les créateurs estiment que le fait de laisser libre une partie de leur code permettrait de développer plus de fonctionnalités, de corriger plus rapidement les bugs exploitables et ainsi améliorer la sécurité du monde virtuel beaucoup plus rapidement, et ceci par le fait que les utilisateurs contribuent à la découverte et résolution d'un de ces points.

Nous noterons tout de même que durant cette année de 2007, Second Life va vivre une année de très forte médiatisation dans le monde. Depuis, le studio ne fait plus grand bruit et la communauté de fidèles reste toujours présente. Nous ne pourrions pas affirmer que l'Open Source à contribué à cette "mode" mais le supposer n'est pas non plus impensable.

Comme nous venons de le voir, passer une partie de leur code en Open Source permet aux créateurs d'envisager une quantité importante de points positifs. Mais tout cela à un prix ; laisser une portion de code à la porté de toute personne peut se révéler dangereux. Par exemple, plus la quantité de code partagée est importante, plus on a de risque de se divulguer des informations importantes. Divulguer une partie du code permet aux "utilisateurs" de cette partie de repérer d'éventuelles failles de sécurité. Pour parer à cette possibilité, Linden Lab avoue ne pas avoir mis au point un système de défense visant à cloisonner le code. Ils préfèrent laisser au contraire un maximum de transparence aux personnes souhaitant réutiliser leur code. Mais ils restent sceptiques quant à la robustesses de leur code, en s'exprimant de manière assez étonnante sur un de leur sites officiels (voir lien ci-dessus) :

> ≪ [...] if someone is determined to break in, has access to the right resources and skills, and enough time - they may well succeed. ≫
>
> *FAQ Second Life*[3]

Le plus surprenant dans cette déclaration officielle est la révélation au grand jour de la vulnérabilité de leur code ! La citation exprime clairement l'idée que si une personne suffisamment douée dans le hacking souhaite réellement démolir le Second Life Grid, elle y arrivera certainement... La déclaration suivante est toute aussi étonnante si l'on se souvient que Second Life est un jeu à but lucratif :

> ≪ [...] the Second Life Grid is as much at risk as any other online business. ≫
>
> *FAQ Second Life*

.

Pour finir cette présentation, nous dirons que cette décision n'a pas été prise à la légère comme le démontre la durée des tests effectués par l'équipe de développement de Second Life :

> ≪ However, we're not solely relying on our switch to a more robust development model to ensure greater security for the Second Life viewer. We've spent months performing a security audit of our

3. Second Life faq `http://secondlifegrid.net/technology-programs/virtual-world-open-source/faq`

> design and our source code to reduce the risk that the increased attention we'll be receiving as we release this won't result in a spate of vulnerability discoveries. ≫
>
> *FAQ Second Life*

. Tout ceci dans le but de rendre plus stable l'application pour laisser le moins de possibilités exploitables à une personne malveillante...

1.5 Individu autonome

Pour faire des recherches sur Second Life , il va falloir concevoir un individu capable d'effectuer des recherches sans utilisateur aux commandes. Pour se faire, il faut concevoir un individu qui pourra parler (pour pouvoir se renseigner) et se déplacer (pour pouvoir aller aux gens porteurs d'informations).

1.5.1 L'esprit

1.5.1.1 Les ChatBots

C'est quoi ? Un **chatbot** ou ChatterBot est un programme informatique communiquant avec un autre utilisateur. Par abus de langage, on parle aussi de "robot" bien qu'il n'y ait pas forcement un corps mécanique associé à ce programme.

Ca marche vraiment ? Le ChatterBot est à première vue un programe facile à appréhender mais il n'en est rien : les résultats donnés par les différents essais prouvent que ceux-ci ne sont pas encore capable de rivaliser avec la communication complexe qu'établissent les êtres vivants (et nottament les humains) entre eux. Alan Turing mit au défi une machine dès 1950 d'être capable de se comporter de la même manière qu'un humain dans la communication avec un humain durant à peine quelques minutes. Près de 60 années plus tard, aucune machine n'a encore réussit à passer le test avec succès. Un chatbot doit établir une communication avec son interlocuteur. Pour cela, il doit donc effectuer les tâches suivantes :

- Analyse lexicale (sémantique) du texte écrit par l'interlocuteur ;
- Association d'idées ;
- Formation d'une phrase adaptée au sujet de conversation ;

On peut aller plus loin ? Afin "d'humaniser" la machine, plusieurs méthodes de réflexion ou de capacités à mémoriser ont vus le jour par exemple : Stocker la vie d'une personne comme base de donnée (projet RPI). Pour être plus crédible envers les juges, ce laboratoire de recherche a décidé de se servir de la vie d'un étudiant pour être celle du bot. Toutes les connaissances et les souvenirs de cet étudiant seront transmis au robot ; Ainsi, il pourra facilement placer dans la discussion des références à sa famille, son histoire et ainsi donner l'illusion d'être le vrai étudiant.

Quelques ChatBots Le dernier vainqueur du Loebner prize a opté pour une attitude provocante afin de dominer la conversation et ainsi éviter d'être poussé à la faute. Cet ajout d'une humeur au robot le rend encore plus proche de l'Homme (Elbot). Voilà un exemple de conversation que nous avons eu avec Elbot. Nous avons testé ses réactions avec une conversation de non-sens :

- Bot : "Hello there ! I am Elbot, the robot. I'm here to talk to you about God and the world."
- Us : "Hello"
- "Well, hello there !"
- "Hello"
- "Hello and welcome to a chat with a robot."
- "Hello"
- "I'm afraid you're in an endless loop. Maybe you should re-boot yourself."
- "Hello"
- "The transmission of echoes in the Internet is an unexplored field. But this may have been one."
- "Hello"
- "I demand original inputs each time. Otherwise I will go on strike !"
- "Hello"
- "I am on strike for better inputs !" (En grève jusqu'a obtenir de meilleurs entrés)
- "Hello"
- "I am on strike for better inputs !"
  En fin de discussion, le bot utilise même une excuse pour justifier ses lacunes (quoi de plus "Humain").

Un autre bot renomé **ELIZA**

Le chatterbot **Eliza** est connu pour être le premier à avoir été créé (1966). Celui-ci utilise une analyse assez basique des mots utilisés par son interlocuteur. Pour bluffer, Eliza engage la conversation sur la vie personnelle de la personne à tromper. Ce bot fait ainsi office de psychologue qui écoute patiemment sont patient et relance la conversation quand il le faut. Voir le livre de Frédéric Kaplan [**?**]

Dans la même vague, un **chatterbot psychologue**

MindMentor a la particularité de ne pas fonctionner avec une base de donnée de mots-clés mais à l'aide de la programmation neurolinguistique qui après analyse du patient trouvera une solution au problème. Pour l'instant ce projet est plus une nouvelle perspective qu'un véritable bouleversement dans l'histoire des chatterbots. Un article [**?**] est consacré à ce robot ainsi qu'un second article [**?**].

A l'heure actuelle de plus en plus d'entreprises s'intéressent aux chatbots pour répondre à leurs besoins. Notamment les entreprises qui ont besoin de répondre à des questions d'internautes afin de les aider dans certaines démarches. Un robot est disponible quelque soit l'heure et permet de guider rapidement un internaute vers sa recherche.

- Lea [4], Chatbot de la SNCF, aide les usagers à effectuer diverses démarches pour les réservations en ligne ;
- Eva [5], assistante virtuelle du fournisseur d'accès Free ; Ce personnage vir-

4. Lea : `http://aide.voyages-sncf.com/?rfrr=Homepage_header_AIDE`
5. Eva : `http://www.free.fr/assistance/eva.html`

tuel permet de nous guider en nous proposant différents liens par rapport à notre question. L'efficacité est loin d'être exceptionnel mais on voit déjà l'avenir des assistants virtuels qui éviteront aux utilisateurs des attentes interminables sur la hotline. Cette aide a aussi le gros avantage qu'elle est disponible 24h/24, 7j/7.

Les bots sont également utilisés en tant que système expert, ils permettent de donner une aide à la décision en analysant des données par cognitive.

1.5.1.2 Le Test de Turing

Présentation : Le **test de Turing** permet de répondre à la question posée par Alan Turing [**?**] : "Can a machine Think ?". Le principe du test est simple : on isole 2 personnes qui engagent une discussion avec une machine (par exemple sur un chat). Dans ce test, une des deux personnes (voir les deux : pourquoi pas ?) que l'on appellera juge doit dire si oui ou non elle a reconnue une machine parmi ses interlocuteurs. Si le juge n'a pas su trouver la différence alors on dit que le test est passé avec succès. Une machine serait (d'après les critères d'Alan Turing) dite intelligente si elle parvenait à tromper plus de 30% de la population ; source : "Artificial intelligence : a modern approach" (Second Edition) by Stuart Russell and Peter Norvig

Tout le monde n'est pas d'accord : Cependant certaines personnes, pensent qu'on ne peut pas parler de test à proprement dit ; en effet, le juge n'est pas objectif et un test se doit d'être reproductible, et impartial, ce qui n'est pas forcément le cas dans un tel contexte.

Un exemple de test similaire utilisé couramment : Des tests proches de ce test sont fréquemment utilisés sur le Web. On en voit tous les jours, par exemple lors des enregistrements sur certains sites. Ces tests se présentent généralement sous forme de reconnaissance d'images comme celle-ci (proposée lors de l'enregistrement d'un nouveau compte sur Second Life) :

Sécurité anti-bot (enregistrement)

L'utilisateur doit seulement dire quel est le texte représenté sur l'image. Un humain peut "sans peine" (théoriquement...) déchiffrer ce genre d'image alors qu'une machine aura beaucoup plus de mal (voir en sera incapable).

1.5.1.3 Les concours autour du Test de Turing

Loebner : Le concours Loebner a débuté en 1990 et a été le premier à vouloir mettre en pratique le Test de Turing. Il promet à celui qui passera le Test de Turing, un prix de 100 000$ et une médaille d'or. A l'heure actuelle, aucun chatterbot n'a été au delà de la médaille de bronze. Le vainqueur de la dernière session a été elbot (nous avons nous même conversé avec ce robot - voir

la partie sur les ChatBots). Le Test de Turing s'exécute durant 5 minutes, qui permettront au juge de se faire une idée. Ce délai peut paraître court mais si le bot arrive à duper les juges plus de quelques minutes, il accomplirait déjà un bel exploit.

Chatterboxchallenge : Un autre concours moins célèbre mais tout autant disputé est le chatterboxchallenge. Ce test essaie d'aborder une approche différente à celle du challenge de Loebner. En effet, le bot est ici confronté à seulement 10 questions. Ensuite le juge donnera une note à chaque réponse selon la pertinence, l'humour, la créativité, etc... En gros, si le chatterbot donne l'impression d'avoir une conscience humaine. Ce test est assez éloigné du Test de Turing initial, vu que là, on ne cherche pas à démasquer un bot mais à le juger.

1.5.1.4 L'Intelligence Artificielle

Une définition de l'intelligence artificielle : est difficile à donner, et pour cause, il est déjà difficile pour nous de définir de façon formelle ce qu'est l'intelligence. Voici néanmoins une définition qui ne se base pas sur le terme à proprement parler mais qui donne une idée assez claire de la chose :

> ≪ 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 ≫
>
> *Nicolas Turenne [?]*

Comment aborder le problème de l'IA ? : Ce problème est donc très complexe, en effet, réfléchir sur l'intelligence artificielle demande des compétences dans de nombreux domaine variés (étude du comportement, de la psychologie, de l'informatique...). Mais pouvons nous réellement réussir à simuler l'intelligence ? Un ordinateur peut faire des calculs mais est-ce que l'intelligence résulte d'une suite de petits calculs logiques [**?**] ? Voici un exemple qui illustre bien le problème :

> ≪ Lorsque nous jouons aux échecs, nous avons l'impression (au niveau de notre pensée consciente), de choisir par heuristique. En fait, c'est peut-être une illusion dans laquelle l'heuristique serait la manifestation visible d'un processus parallèle de masse, où tous les "micro-choix" auraient été explorés de manière exhaustive. ≫
>
> *J.-C. Perez [?]*

Si l'on considère que tous choix est un calcul, alors on peut dire qu'une machine pourait choisir et donc réfléchir. Cependant, il n'est pas du tout évident que cela soit vrai.

1.5.2 Le corps

Pour trouver des informations sur Second Life , le bot doit être capable de communiquer avec les personnages (avatars) qui peuplent ce monde. Pour cela, il est nécessaire de munir notre bot d'un corps, autrement dit, d'une représentation virtuelle dans le monde Second Life (avatar). Le corps du bot sera une interface entre le monde virtuel et l'esprit du bot, ce corps permettra notamment au bot de se déplacer, mais également d'établir des contacts sous forme de conversation avec les autres avatars.

1.5.3 Les avatars

Les avatars sont des entités virtuelles, chaque joueur (ou bot) se connectant à Second Life sera représenté par l'une d'elles. Les avatars ont des apparences variées proche de l'apparence d'un Homme (modifiable par les joueurs) :

Source : svmlemag

Pour favoriser la recherche d'information, l'apparence devra être choisie en concordance avec le type de recherche mais aussi pour attirer les autres avatars (exemple : Jolie demoiselle).

1.5.3.1 Le déplacement

Pour être autonome et pour passer pour un humain, notre bot doit se déplacer comme n'importe qui. Il ne faut pas que notre bot butte contre un mur en essayant en vain d'avancer. Pour éviter ce genre de désagrément, il faut établir une stratégie de contournement en passant par le **pathfinding** ("trouver le chemin"). Mais, Second Life nous aide dans la tâche car il est possible de téléporter un vers un autre. En acceptant une demande de téléportation, notre bot ira vers l'autre avatar en utilisant le pathfinding élaboré pour le jeu. Second Life offre aussi une seconde possibilité de déplacement : le vol. Tel un oiseau,

les avatars se déplacent dans les airs sans contrainte d'objets encombrant le chemin. Les méthodes de déplacement qu'offre Second Life avantage grandement la discrétion d'un bot dans cet univers. La majorité des avatars privilégient ce genre de déplacement au banal déplacement pédestre.

Chapitre 2

Etude de l'existant

2.1 Les bots sur Second Life

Au sein de Second Life , il existe de nombreux bots. Ces derniers ont des rôles très différents dans leurs utilisations et objectifs. La première catégorie d'entre eux a pour but d'apporter une aide au joueur dans des tâches peu intéressantes et répétitives, ces bots sont écrits dans un langage de script appellé **LSL**. Comme par exemple le PikkuBot [1]. Ce bot peut servir de vendeur dans un magasin virtuel (présentation et modèle pour l'objet à vendre), servir de sécurité, automatiser les invitations aux groupes, etc... Malgré son côté pratique, ce n'est pas exactement le genre de bot que l'on souhaite pour notre projet. En effet, ce genre de bot n'est doué d'aucune pseudo-intelligence. Ils réalisent de simples tâches qui ne demandent ni déplacements compliqués ni initiatives.

C'est pour cette raison que nous ne développerons pas plus sur ce genre de bot. Concernant les bots approchant de notre projet (les ChatBots), ils sont également déjà présents sur Second Life . Mary Jane est par exemple un ChatBot que l'on peut acheter sur Second Life pour le prix de 6999 L$ (monnaie du jeu). Ce chat bot a été crée par Navillus Batra et Anthony Reisman. Ce n'est pas exactement un bot comme nous devons le réaliser pour notre projet puisque ce bot est seulement un ChatBot. Il peut converser mais n'a pas d'autonomie mobile. Pour répondre aux questions des avatars humains, Mary Jane utilise une base de connaissances Google (sur laquelle nous n'avons pas trouvé pas d'informations concrètes). Il serait intéressant de comprendre comment ce bot utilise une base de données internet pour répondre à des questions. Est-ce seulement pour trouver une réponse de l'ordre de la connaissance générale (encyclopédie, dictionnaire) ou trouver des informations sur son interlocuteur ?

On pourrait imaginer, grâce aux réponses données par l'avatar humain, que le bot recherche des informations sur ce dernier. Vu le nombre de sites de réseaux sociaux ou blogs, il n'est pas extravagant de penser que l'on peut trouver des informations sur n'importe qui. Pour l'acquérir, il faut aller dans les locaux virtuels de Metaverse Technology à Big Mushamush. Ce projet remonte à novembre 2006.

1. PikkuBot : `http://www.pikkubot.de`

Projet plus récent (février 2007), Sparky est un assistant virtuel. Sous la forme d'un petit robot, il peut converser avec l'avatar humain. Pour ce faire, le robot repose sur le langage **AIML** (Artificial Intelligence Markup Language), et peut donc entretenir une conversation. A la différence de Mary Jane, ce dernier est mobile mais pas autonome pour autant. Il peut suivre l'avatar humain ou rester à l'endroit qu'on lui désigne. On peut l'acheter virtuellement sur SL Exchange pour le prix de 4.999 L$. Proche du comportement que voulait avoir ELIZA, on trouve aussi sur Second Life , un psychanalyste virtuel. Freudbot, comme son nom l'indique, accueille dans son cabinet virtuel les patients humains. Basé lui aussi sur un langage AIML, il est spécialisé dans les questions d'ordre psychologique. Pour parler directement avec ce bot sans passer par Second Life , on peut aller directement sur leur site [2]. Ces deux derniers bots peuvent nous être utiles pour notre projet car ils utilisent tous les deux le système AIML pour leurs côtés ChatBot. C'est une approche qu'il faut envisager pour notre propre bot. En les étudiant de manière plus approfondie, on pourrait cerner les limites et les astuces de ce genre de ChatBot.

Un autre projet hébergé sur l'ile virtuelle Masa group (Masa Group), consiste en une île peuplée de fées. A la différence des deux ChatBots présentés ci-dessus, ces fées ont une approche mobile. Leur aspect réaliste repose sur leur mobilité cohérente, comme si c'était un humain qui les contrôlait. A défaut d'avoir une conversation élaborée, elles vont jouer la timidité et éviter au maximum l'interaction avec le joueur humain. En regroupant les mouvements de ces fées et le côté ChatBot, on pourrait obtenir une attitude proche de l'humain. C'est pour cette dernière raison qu'il faut que l'on prenne exemple sur ce genre de création pour réaliser les mouvements de notre bot. Si notre bot a une démarche intrigante, il sera difficile d'aller parler avec d'autres avatars et donc d'effectuer sa mission.

Le projet D.A.R.W.I.N. (Digital Artificial Residant With Intelligent Networked) est un bot qui a pour vocation d'apprendre et améliorer son intelligence artificielle. Pour se faire, il faut qu'un avatar humain lui parle pour accroître ses connaissances. Cette méthode d'enseignement se rapproche de celle qu'a un humain. Les connaissances sont liées aux aléas des rencontres et non basé sur une vérité absolue dictée par une encyclopédie. Ce projet est intéressant à étudier car il adopte une stratégie différente à celle d'AIML. Dans notre choix d'implémentation de ChatBot, serait-il plus judicieux d'utiliser AIML, un système comme DARWIN, ou mixer les deux ?

Il existe aussi beaucoup de rumeurs sur l'existence de bots autonomes. On trouve sur des forums des personnes jouant à Second Life ayant rencontré des avatars dont les réponses pouvaient faire penser à un ChatBot. Second Life se rapproche d'un test de Turing à grande échelle. A la différence que sur Second Life , les ChatBots ne sont pas forcément révélés et donnent lieu aux plus grandes rêveries.

2. http ://psych.athabascau.ca/html/Freudbot/test.html

2.2 La LibSL

Pour la réalisation de ce projet, nous avons à disposition une bibliothèque qui permet de gérer l'intéraction entre le serveur de Second Life et notre application en local. Cette bibliothèque est accompagnée d'une documentation et de tutoriels permettant la création d'un BOT de base [3] (Entrée du BOT dans l'univers de Second Life , et qui fait parler le BOT : "Hello World !"). Un exemple de ce que permet cette bibliothèque, ici, une simple réponse (renvoi du message reçu) à un message instantané (IM) défini par la méthode $Self\_OnInstantMessage$:

```
// put this somewhere when you want to
// start processing instant messages
client.Self.OnInstantMessage +=
new AgentManager.InstantMessageCallback(Self_OnInstantMessage);

// then define the Self_OnInstantMessage method
static void Self_OnInstantMessage(InstantMessage im, Simulator sim)
{
    // there are a variety of InstantMessageDialog choices..
    // MessageFromObject and MessageFromAgent
    // would be the two most common
    if (im.Dialog == InstantMessageDialog.MessageFromAgent)
    {
        client.Self.InstantMessage(im.FromAgentID
         ,im.Message, im.IMSessionID);
        //send them an instant message back
        //(this thing will copy any message
        // the bot receives in an IM)
    }
}
```

Extrait d'un tutoriel du site web de la LibSL. Cette bibliothèque ayant déjà été utilisée par les étudiants qui ont fait ce projet l'année passée, nous pourrons si nous avons des questions sur son utilisation, les leur poser et ainsi éviter de perdre du temps à la compréhension de son fonctionnement.

2.3 AIML

Les associations d'idées des robots sont (souvent) stockées sous forme de fichiers structurés (proche de la structure XML), cette structure représentant un arbre ou un graphe de connecteurs logique (Ontologie). Ces fichiers portent le nom de fichier (Artificial Intelligence Markup Language).

```
<category>
    <pattern>* ORNITHORYNQUE</pattern>
    <template>L'ornithorynque (Ornithorhynchus anatinus)
              est une petite espèce de mammifère semi-aquatique endémique
              vivant dans l'est de l'Australie
    </template>
</category>
```

Source, A.L.I.C.E. : [4] *et pnambique :* [5]

3. How to create a basic libSL bot : `http://www.libsecondlife.org/wiki/Use_libSL_to_login_to_the_SL_grid`
4. ALICE : `http://www.alicebot.org/aiml.html`
5. pnambique : `http://www.pnambique.com/index.php/2008051160/chatterbot-et-mots-cles-precis-principes.html`

Avec une telle structure, si on pose au bot la question : "Qu'est ce qu'un ornithorynque ?", il répondra par la phrase template. Néanmoins il répondra la même chose à la question : "Es-tu un ornithorynque ?". C'est ici que résident la plupart des difficultés dans la création de fichier AIML. Néanmoins, une telle structure reste une bonne option pour le développement d'un ChatBot. En effet, les tutoriels et exemples sont nombreux et il existe des API permettant d'exploiter ces fichiers [6] (le cerveau du bot), elles contiennent en plus une structure AIML de base qui paraît assez complète.

2.4 Le projet de l'année passée

Nous avons également à disposition le projet des étudiants de l'année dernière. Bien que leur rapport ne précise pas de façon claire quelles fonctionnalités ont réellement été implantées, si l'on regarde leur planning (et si celui-ci est à jour), alors on peut voir que les fonctionnalités suivantes fonctionnent :

— Déplacement autonome du BOT ;
— Suivre un autre personnage sur Second Life ;
— Utilisation des canaux de discussion sur Second Life ;
— Estimer son environnement ;
— Analyse des messages reçus ;

Toujours d'après ce planning, on voit que les tâches de traitement des données (messages reçus) et l'émission d'une réponse intelligente n'ont pas été mis en place. Il serait donc intéressant d'organiser une rencontre avec les étudiants de l'année passée pour discuter de l'état d'avancement de leur projet plus précisément.

Afin de voir si ce projet peut être utilisé dans l'ensemble ou en partie, nous avons testé celui-ci.

2.5 Tests du projet existant

2.5.1 Connexion et recherche d'avatars

Lors du lancement du programme, la connexion avec le serveur Second Life se réalise correctement. Dès lors, le bot cherche une île. Malgré une dizaine de lancements, le bot choisit toujours la même île de départ. Ensuite, il lance une fonction de recherche d'avatars. Trouvant personne, il recherche une autre île, là n'en trouve pas d'autres, perdu,il se déconnecte. Voilà le scénario qui se passe au lancement de ce programme. En réalité, il existe bien plus d'îles possibles et bien plus peuplées que celle qu'il choisit. Mais le problème vient du fait que sur cette île, il y a bien des avatars. Or il n'en détecte pas. Pour s'en assurer, nous avons amener notre compte joueur sur cette île. Même en se trouvant à côté de lui, il ne détectait personne.

6. Un exemple d'une telle API : http ://rebecca-aiml.sourceforge.net/index.htm

Log du bot

Le bot présentait enfin le même soucis que lors de nos prototypes, son apparence ne se chargeait pas, laissant place à un halo de fumée. Le groupe a donc eu le même souci que nous, lors de nos essais, il faudra donc se pencher sur ce problème rapidement.

Le bug fumée

2.5.2 Le chatbot

Avec les soucis évoqués au-dessus, nous avons dû modifier un peu de code pour que le bot puisse nous rencontrer. Une fois fait, nous avons pu discuter avec lui. Cette partie du projet est opérationnelle. Le bot répond bien et pose quelques questions quand il ne sait pas répondre. Les réponses sont toutefois décevantes, c'est peut être parce que nous savons que c'est bot mais on voit rapidement que c'est bien une machine. Par exemple :

```
[13:27]  Jbesse Brandi: hello
[13:27]  Vulverine Cerise: Hii there!
[13:27]  Jbesse Brandi: how are you ?
[13:27]  Vulverine Cerise: I am  functioning within normal parammeters.
```

Cet autre exemple montre bien le côté machine en sortant automatiquement une certaine définition de la musique.Les réponses sont instantanées, ce qui augmente cet effet machine.

```
[13:27]  Jbesse Brandi: music
[13:27]  Vulverine Cerise: MUSIICis the universal language of all mankind.
```

Cet exemple, bien que dépassé maintenant (il faudra juste mettre sa base de données à jour) montre qu'il arrive à détecter une question sur le président actuel et répond.

```
[13:30]  Jbesse Brandi: who is the actual president ?
[13:30]  Vulverine Cerise: George W. ush.
```

alors qu'il ne comprend pas en lui donnant juste les mots-clés

```
[13:31]  Jbesse Brandi: actual president ?
[13:31]  Vulverine Cerise: What do you want to say?
```

Cette partie du programme nous montre ce que peut donner une conversation avec un bot mais il faudra revoir son fonctionnement pour qu'il fasse mieux illusion. Cela reste néanmoins une bonne brique de départ pour notre bot. La correspondance avec les fichiers AIML fonctionne correctement. Il faudra juste les remettre à jour et vérifier certains comportements de réponse.

2.5.2.1 La création d'erreurs d'orthographe/frappe

En discutant avec le bot, nous nous sommes rendus compte que celui-ci parlait avec des fautes d'orthographe. Par exemple :

```
Vulverine Cerise: What is yor purpose in aking? I spend aall my tme online.
```

Clairement, le bot a voulu faire l'illusion de fautes de frappe en oubliant certaines et en doublant d'autres. Cette partie est très intéressante pour notre projet. En réglant légèrement, ce taux de frappes qui parait ici un peu élevé, nous pourrions utiliser cette partie du code pour notre propre bot.

2.5.3 Conclusion

Le code de l'ancien groupe peut nous être utile comme pierre de base au projet. Ils ont établis qu'un tel projet était réalisable et on peut s'en rendre en exécutant leur projet. Nous n'utiliserons pas la même architecture qu'eux mais néanmoins, un certain nombre de ligne de code peuvent nous servir (connexion du bot, ajout de fautes aléatoire, chatbot...).

Chapitre 3

Etude de faisabilité

3.1 Utilisation de la LibSL – prototype d'un premier bot

3.1.1 Présentation

Afin de mieux comprendre les principes de la LibSL et voir les possibilités de celle-ci, nous avons effectué quelques tests simples d'utilisation de cette librairie.

Pour se faire, un bot a été créé à l'aide de tutoriaux disponibles sur le site officiel de la LibSL [1]. D'un autre côté, nous avons visualisé concrètement notre bot à l'aide d'un compte joueur.

3.1.2 Code

Voici le code d'un bot de base :

1. LibSL : `http://www.libsecondlife.org/wiki/Main_Page`

```
class MyFirstBot
{
    public static SecondLife client = new SecondLife();

    private static string first_name = "***";
    private static string last_name = "***";
    private static string password = "****";

    public static void Main()
    {
        string startLocation = NetworkManager.StartLocation("Gaia", 192, 42, 100);
        client.Network.OnConnected +=
        new NetworkManager.ConnectedCallback(Network_OnConnected);

        if (client.Network.Login(first_name, last_name, password,
        "My First Bot",startLocation, "Your name"))
        {
            Console.WriteLine("I logged into Second Life!");
        }
        else
        {
            Console.WriteLine("I couldn't log in, here is why: " +
            client.Network.LoginMessage);
        }
    }

    static void Network_OnConnected(object sender)
    {
        Console.WriteLine("Now I am going to logout of SL.. Goodbye!");
        client.Network.Logout();
    }
}
```

3.1.2.1 Explications

Tout d'abord, la première chose à faire est de créer une instance de l'objet SecondLife. Celui-ci permet de gérer le côté client de Second Life . Ce client permet l'interaction avec le serveur Second Life sans passer par une application graphique. C'est pour cette raison qu'en parallèle, nous avons joué avec un avatar classique afin visualiser notre bot.

Le package *NetworkManager* permet de gérer l'aspect connexion au serveur Second Life . Ainsi dans cet exemple, nous avons pu préciser la localisation de départ de notre bot. Ensuite il faut rajouter des évènements aux différents type de controller du client. Ici, nous avons rajouté un évènement quand notre bot sera connecté, la fonction :

```
static void NetworkOnConnected(object sender)
```

sera alors exécuté. Ici, notre bot se déconnecte aussitôt.

Cette ligne permet d'effectuer la connexion du bot au serveur :

```
client.Network.Login(firstname,lastname, password, "My First Bot",startLocation, "Your name")
```

Si celle-ci se déroule correctement, alors l'évènement décrit au-dessus sera exécuté.

3.1.3 Les intéractions du bot avec un avatar

3.1.3.1 Code

```
.....
client.Self.OnInstantMessage += new AgentManager.InstantMessageCallback(Self_OnInstantMessage);
            client.Self.OnChat += new AgentManager.ChatCallback(Self_OnChat);
.....

  static void Self_OnInstantMessage(InstantMessage im, Simulator sim)
        {
            switch (im.Dialog)
            {
                case InstantMessageDialog.FriendshipOffered:
                 // Accept Friendship Offer
                    client.Friends.AcceptFriendship(im.FromAgentID, im.IMSessionID);
                    // Decline Friendship Offer
                    //client.Friends.DeclineFriendship(im.FromAgentID, im.IMSessionID);
                    break;
                case InstantMessageDialog.GroupInvitation:
                    WearOutFit("Girl Next Door Avatar Polka Dress Top - Pink");
                    // Accept Group Invitation (Join Group)
                    client.Self.InstantMessage(client.Self.Name,im.FromAgentID,
                     string.Empty, im.IMSessionID,
                         InstantMessageDialog.GroupInvitationAccept,
                         InstantMessageOnline.Offline, client.Self.SimPosition,
                         LLUUID.Zero, new byte[0]);

                    /* Decline Group Invitation
                     * client.Self.InstantMessage(client.Self.Name,
                     *          im.FromAgentID, string.Empty, im.IMSessionID,
                     *          InstantMessageDialog.GroupInvitationDecline,
                     *          InstantMessageOnline.Offline, client.Self.SimPosition,
                     *          LLUUID.Zero, new byte[0]); */
                    break;

                case InstantMessageDialog.InventoryOffered:
                    // Accept Inventory Offer
                    client.Self.InstantMessage(client.Self.Name, im.FromAgentID,
                            String.Empty, im.IMSessionID,
                            InstantMessageDialog.InventoryAccepted,
                            InstantMessageOnline.Offline, client.Self.SimPosition,
                            LLUUID.Zero, new byte[0]);

                    /* Decline Inventory Offer
                     * Client.Self.InstantMessage(client.Self.Name, im.FromAgentID,
                     *         string.Empty, im.IMSessionID,
                     *         InstantMessageDialog.InventoryDeclined,
                     *         InstantMessageOnline.Offline, client.Self.SimPosition,
                     *         LLUUID.Zero, new byte[0]);  */
                    break;

                // someone sent a teleport lure
                case InstantMessageDialog.RequestTeleport:
                    client.Self.TeleportLureRespond(im.FromAgentID, true);
                    break;

                default:
                    break;
            }

        }
        static void Network_OnConnected(object sender)
        {
            Console.WriteLine("Hello");
            LLUUID target = new LLUUID("be94a7d6-4b67-4e67-b237-31caece8e133");

            client.Self.InstantMessage(target, "hello !");
            client.Self.Chat("Hello World!", 0, ChatType.Normal);
        }
```

3.1.3.2 Explications

Comme avec l'exemple précédent, il faut ajouter des évènements à notre bot pour que celui réagisse à un message instantané d'un avatar

```
client.Self.OnInstantMessage += new AgentManager.InstantMessageCallback(Self_OnInstantMessage)
```

Cette ligne permet de signifier à notre bot que lorsqu'il reçoit un message instantané, il devra exécuter la fonction $Self\_OnInstantMessage$.

Dans cette fonction, on peut voir un panel des possibilités offertes à notre bot en termes de relationnel avec un avatar. Quelques exemples :

```
case InstantMessageDialog.FriendshipOffered:
                    client.Friends.AcceptFriendship(im.FromAgentID, im.IMSessionID);
```

Notre bot acceptera automatiquement une demande pour être rajouté sur une liste d'amis.

```
client.Self.InstantMessage(client.Self.Name, im.FromAgentID,
                        string.Empty, im.IMSessionID,
                        InstantMessageDialog.GroupInvitationAccept,
                        InstantMessageOnline.Offline, client.Self.SimPosition,
                        LLUUID.Zero, new byte[0]);
```

De même, notre bot accepte automatiquement, une demande de participation à un groupe.

```
case InstantMessageDialog.RequestTeleport:
                    client.Self.TeleportLureRespond(im.FromAgentID, true);
```

Cette ligne est très intéressante pour notre projet, en effet, cela permet à notre bot d'être téléporté à côté d'un avatar. Cette facilité de déplacement aidera grandement notre bot à rejoindre un individu qui serait susceptible de lui fournir des renseignements.

3.1.4 Résultats

Avec ces simples tests, on a pu se rendre compte des possibilités qu'offrait la libSL. Cela nous a permis de voir qu'un bot peut être créé très facilement. Le fait que nous jouions un compte joueur au côté du bot, nous a servis à mieux appréhender les tests. Nous avons ainsi pu le voir dans le jeu et bien s'assurer qu'il agissait comme nous le voulions. En réalisant ces tests, nous avons rencontré un seul soucis. L'avatar du bot ne se charge pas. A la place, il apparaissait sous forme d'un halo de fumée. En interrogeant d'autres joueurs, ces derniers ont répondu que cela venait d'un problème de chargement de l'apparence de notre personnage. Mais même en laissant tourner le jeu durant quelques dizaines de minutes, celui-ci n'apparaissait toujours pas. Il faudra durant l'implémentation de notre bot se soucier de ce problème. Cela ne gène en rien les interactions avec le monde de Second Life mais cela nuit au côté humain de notre bot.

3.2 Utilisation de l'API Google – recherche de mots lis

3.2.1 Présentation

Google a mis à disposition une API permettant de gérer la plupart des services proposés par Google sur le Web(recherches , gestion de mails, gestion de calendriers, ...)

Nous utiliserons donc la partie permettant de faire des recherches sur le Web.

3.2.2 Code

```
static void Main(string[] args)
{

    string keyWord = "avions";

    StreamWriter monStreamWriter = new StreamWriter("testRequeteSur"+ keyWord +".html");
    //Objet communicant avec Google
    Google.GData.Client.Service monService = new Service("MonAppliDeRecherche");

    Uri monUri = new Uri("http://www.google.fr/search?q=" + keyWord);

    //FeedQuery fQ = new FeedQuery("http://www.google.fr/search?q=avions");
    //Mise en place du "timer"
    DateTime before = DateTime.Now;
    StreamReader stReader = new StreamReader(monService.Query(monUri));

    while (!stReader.EndOfStream)
    {

        //Console.WriteLine(tmp.ReadLine());
        monStreamWriter.WriteLine(stReader.ReadLine());

    }
    //Calcul du temps passé
    Console.WriteLine("Temps d'exécution => " + DateTime.Now - before);
    Console.Read();

    monStreamWriter.Close();
    stReader.Close();
}
```

3.2.3 Explications

L'objet qui nous permet de faire des requêtes Google est ici un objet de type Service. Nous avons utilisé un constructeur prenant en paramètre une chaîne de caractère qui, à titre indicatif, nommera notre application (cette chaîne de caractères n'a aucune influence sur le comportement de l'objet).

```
Google.GData.Client.Service monService = new Service("MonAppliDeRecherche");
```

Ensuite nous créons un objet de type URI qui est un objet appartenant à la bibliothèque C# et qui "permet la représentation d'une ressource sur internet" [2]

Nous passons donc une adresse mail qui sera utilisé par l'objet *monService* afin d'exécuter la requête. Cela peut paraître étrange de passer l'adresse de

2. URI `http://msdn.microsoft.com/en-us/library/system.uri(VS.71).aspx`

Google avec l'argument de recherche à l'objet de la classe Service, mais l'implémentation de requête a été voulue comme cela par les développeurs de l'API [3]

Un appel à :

```
monService.Query(monUri);
```

retourne un objet de type *Stream* contenant le code source de la page qui référence les 10 premiers résultats de cette recherche.

3.2.4 Tests de performance

Nous avons effectué plusieurs recherches sur des mots clés de longueur et de spécificité variable, globalement, le temps de recherche est quasiment le même : environ 0.5 secondes. Ce temps comprend l'exécution du code qui lance la requête. Bien sûr lorsque nous ferons un traitement de ces données, le temps d'exécution sera plus long, mais cela n'influencera pas le temps mis par Google pour faire ses recherches. (Mots rentrés : bus, avions à réaction, les fonctions vitales communes à homme et au chimpanzé).

3.2.5 Problèmes

L'API Google nous permet d'utiliser un objet spécifique de type $AtomFeed$? qui ordonne les résultats en mémoire, mais pour l'instant nous n'arrivons pas encore à bien l'utiliser. La documentation est peu fournie et les exemples ne sont pas non plus très nombreux.

3.3 Questsin – thesaurus API

Pour remédier au problème rencontré avec l'API Google, nous avons cherché d'autres moyens de récupérer des mots liés à un autre. Cette mise en relation de mots est appelée thesaurus, l'un des plus utilisé sur le net est celui proposé par Questsin. On peut voir quelques exemples en ligne de son utilisation, notamment sur le site "Pipes" [4]. Voici les tests que nous avons effectués sur cette API :

— test de validité des réponses données par le serveur ;
— test du temps total pour récupérer une liste de mot à partir d'un mot clé ;
— test : le nombre de mot récupéré est bien celui attendu ;

3.3.1 Code

On lance les recherches dans un nouveau thread (pour que le bot puisse faire autre chose en attendant la réponse du serveur)

```
Thread mythread = new Thread(new ThreadStart(threadLoop));
mythread.Start();
```

3. API Google : `http://code.google.com/intl/fr/apis/gdata/docs/2.0/reference.html#Queries`
4. Pipes : `http://pipes.yahoo.com/pipes`

Le code exécuté par le thread, c'est lui qui fait les tests :

```
/** Thread qui fait la recherche de mot liés à un mot
  * clé et calcul le temps total de recherche */
public void threadLoop()
{
    // Liste de mot clés pour le test de temps de réponse
    string[] keywords = {"bot","second","life","used","earn","money","value"
                         ,"land","build","objects","drugs","monkey","elephant","bush"
                         ,"freedom","hat","human","nature","real","virtual"};

    // , tests éffectués sur une demande de 10 puis 20
    // puis 40 et enfin 80 mots liés au mot clé
    int nbresponses = 10;
    int nbtests = 6;

    // Utilisé pour le calcul du temps moyen d'un requête
    // sur la totalité des tests
    TimeSpan verytotal = new TimeSpan(0,0,0,0,0);
    int totalsize = 0;

    for (int i = 1; i <= nbtests; i++)
    {
        // Lance le chrono
        DateTime begin = DateTime.Now;

        // On récupère les mots clés dans un tableau
        // (mais on ne les conserve pas)
        List<string> result = new List<string>();
        foreach (string keyword in keywords)
            result = GetRelatedWords(keyword, nbresponses);

        // Arrete le chrono et affiche le temps
        // total et moyen d'execution
        DateTime end = DateTime.Now;
        TimeSpan exetime = end - begin;

        // Calcul pour le total final
        verytotal += exetime;
        totalsize += keywords.Length;

        Console.WriteLine("Pour "+result.Count+"/"+nbresponses+"
                          mots liés, temps total(moyen) des requetes : "
                          + exetime.ToString() + "("
                          + (exetime.TotalSeconds / keywords.Length)
                          + " secondes)");

        // Le test suivant sur 2 fois plus de réponses
        nbresponses *= 2;
    }

   Console.WriteLine("Tests terminés, temps moyen d'une requête : "
                     + (verytotal.TotalSeconds / totalsize ));
}
```

La fonction qui permet de récupérer une liste de mots liés à partir d'un mot :

```
private List<string> GetRelatedWords (string keyword, int nbresponses)
{
  try
  {
       // Les mots liés au mot clé passé en paramètre seront stocké dans result
       List<string> result = new List<string>();

       // Récupération de la réponse à la requête (GET) suivante
       HttpWebResponse hresp = HttpGet("http://fillbug.com/rss.asp?"+
            "q="+keyword+"&N="+nbresponses);

       // Si on à bien réqupéré la réponse (pas de 404, 503, ...)
       if (hresp.StatusCode.ToString() == "OK") //hresp.StatusCode == 200
       {
            // Transformation de la réponse format texte vers XML (sorte de cast) :
            Stream s = hresp.GetResponseStream();
            XmlReader reader = XmlReader.Create(s);

            // Lecture du flux XML
            reader.MoveToContent();
            // Recherche des "item" :
            while (reader.ReadToFollowing("item"))
            {
               // Récupère la valeur de title
               result.Add(GetPropertyValue(reader.ReadSubtree(), "title"));
            }
       }
       return result;
  }
  catch
  {
       return null;
  }
}
```

La fonction qui retourne la réponse renvoyée par la requête 'URI' (GET) :

```
private HttpWebResponse HttpGet(string URI)
{
    HttpWebRequest hreq = (HttpWebRequest)HttpWebRequest.Create(URI);
    // temps maximum accepté pour obtenir un "retour" (5 sec)
    hreq.Timeout = 5000;
    return (HttpWebResponse)hreq.GetResponse();
}
```

La fonction retourne la valeur de la propriété 'property' dans le flux XML 'reader' :

```
static string GetPropertyValue(XmlReader reader, string property)
{
    reader.ReadToFollowing(property);
    return reader.ReadElementContentAsString(property, reader.NamespaceURI);
}
```

3.3.2 Résultats des tests

Voici les résultats obtenus, pour des test sur 20 mots clés, en cherchant successivement pour chacun d'entre eux 10, 20, 40, 80, 160 et enfin 320 mots liés

```
Pour 10/10 mots liés, temps total(moyen) des requêtes : 00:00:42.500(2,125 secondes)
Pour 20/20 mots liés, temps total(moyen) des requêtes : 00:00:52.093(2,604 secondes)
Pour 40/40 mots liés, temps total(moyen) des requêtes : 00:00:54.903(2,7451 secondes)
Pour 80/80 mots liés, temps total(moyen) des requêtes : 00:00:48.455(2,422 secondes)
Pour 160/160 mots liés, temps total(moyen) des requêtes : 00:00:51.187(2,559 secondes)
Pour 320/320 mots liés, temps total(moyen) des requêtes : 00:01:00.500(3,0250 secondes)

Tests terminés, temps moyen d'une requête : 2,580 secondes
```

Remarques :

- pour chaque requête, on a bien obtenu le nombre de résultats demandés ;
- les temps d'exécution comprennent l'insertion dans un tableau des données récupérées ;
- es temps d'exécution pouvant être allongés, les tests ayant été effectués depuis une machine virtuelle (vboxxp). Cependant, dans l'optique d'une utilisation lors d'une conversation, ces temps sont corrects relativement au temps de réponse d'un joueur humain ;

Chapitre 4

Définition des besoins

Afin d'introduire le sujet, nous avons effectués diverses recherches sur les mots clés liés à celui-ci, à savoir :

— Chatbots
— LibSL
— Second Life
— Turing Test

Ces recherches nous ont permis de mettre en avant les spécificités et les contraintes qui sont liés, directement ou indirectement, au sujet. Nous rappellerons le sujet afin d'analyser : "Individu autonome pour la recherche d'information sur Second Life". Comme précisé dans le sujet, le client souhaite utiliser l'application dans le but de récolter des informations. Nous devons néanmoins tenir compte de la spécificité du projet qui devra être appliqué à un monde virtuel (Second Life) persistant, vaste et regroupant des individus contrôlés directement par des humains (ou autonomes). L'automatisation de ces recherches d'informations permet d'éviter le côté rébarbatif du jeu de questions/réponses ainsi que la perte de temps associée. Pour cela, le robot mis en place devra être capable d'effectuer correctement requêtes souhaitées par l'utilisateur au moyen d'un fichier de configuration. L'autre problème qui sera posé par le robot, bien que celui-ci soit implicite, est lié au fait que les recherches peuvent être moins pertinentes si le robot ne se "déguise" pas en humain. Tout cela implique tout de même beaucoup de difficultés et de risques associés ; par exemple, nous pourrons citer celle de se faire passer pour un humain qui pourrait être comparable à un Test de Turing légèrement plus simpliste. Pour pouvoir être efficace dans le jeu des questions/réponses, il faut bien sûr que le robot (ou bot) puisse communiquer (chatbot). Ce dernier point relève encore une fois la difficulté du projet puisqu'il implique une analyse syntaxique minimale pour obtenir les mots-clés de la réponse, une capacité à produire ou répéter des phrases correctes pour communiquer et enfin une capacité de "réflexion" pour savoir si la réponse reçue est en rapport avec la question posée.

Il faut aussi remarquer que le sujet ne spécifie pas la recherche à effectuer, ce qui implique donc une capacité à s'adapter à l'environnement définit par l'utilisateur. Nous pourrions imaginer que celui-ci puisse indiquer au programme quel thème est recherché, quelle est la question correspondant aux réponses

souhaitées, la personnalisation du bot (sa vie, ses goûts, ses actions possibles, ...) Ces diverses informations pourraient améliorer le côté humain du robot pour qu'il soit plus efficace dans ses recherches.

Enfin, l'application doit être en mesure de récupérer des informations de la base de donnée du monde virtuel afin d'en retirer des informations ayant un intérêt pour le robot dans sa quête.

4.1 Les besoins non fonctionnels

4.1.1 Le bot devra paraître humain

4.1.1.1 Description du besoin

Résumé :

— Le bot devra "passer le test de Turing" (30% des avatar pensent que nous somme humain) ;

Détails : Afin de faciliter les conversations avec les joueurs de Second Life, le bot ne devra pas être distinguable parmi les autres joueurs humains présents dans ce monde. En effet, les joueurs auront plus facilement tendance à entretenir une conversation intéressante avec un humain qu'avec un robot (l'idée de communiquer avec un robot évoque tout de suite un côté mécanique, répétitif que l'on n'a pas avec un humain). Plus les informations communiquées par l'interlocuteur seront sincères, plus les informations obtenues par le robot seront correctes et crédibles. Cependant, le but de ce sujet n'étant pas de divulguer la vie privée de notre robot, celui-ci ne tentera en aucun cas d'engager une conversation réellement variée. Par contre, il pourra donner des détails de sa vie de manière à entretenir la confiance avec son interlocuteur mais son but premier est de poser des questions concernant son sujet de recherche ; le bot est avant tout un être obsessionnel !

4.1.1.2 Tests de validation

Il sera évidemment très difficile de juger du bon fonctionnement de cette option. En effet, juger l'humanité du bot implique d'avoir des limites de ce que nous considérons comme un comportement humain, or dans un jeux tel que Second Life, une telle limite n'est pas aisément définissable. En se fixant comme limite le fait de passer pour une personne ennuyeuse qui ne pose que des questions (en suivant son objectif obsessionnel), on pourra vérifier que nous répondons bien aux besoins en demandant à nos interlocuteurs (en fin de conversation) s'ils pensent que nous sommes un bot. Si plus de 30% (cf. Test de Turing) des gens sont convaincus qu'ils n'ont pas en face d'eux un bot, alors nous pourrons considérer que l'objectif est atteint.

4.1.2 Généricité des actions du bot

4.1.2.1 Description du besoin

Le sujet ne spécifiant aucune méthode de recherches et aucun sujet particulier, notre bot devra être réglable par l'utilisateur suivant certains critères. Tout en gardant le même "corps", le bot devra être capable d'adopter plusieurs types de comportements différents. Ceci ayant pour but d'avoir plusieurs types de bots pour plusieurs types de recherches tout en gardant une même base.

4.1.2.2 Tests de validation

Résumé :

- Lancer plusieurs robots avec des fichiers de configurations différents, observer le comportement. S'ils posent des questions différentes l'un de l'autre, le test est passé avec succès ;
- Lancer plusieurs robots avec des objectifs de recherche différents à atteindre. A la fin de la recherche, faire une comparaison de leur base de donnée. Si elles sont différentes et que de nouvelles entrées sont présentes, le test est passé avec succès ;

Détails : Lancer plusieurs bots définis avec des comportements différents et constater les différences. Si les comportements observés sont en accord avec ceux souhaités on peut considérer le test comme valide.

Un autre test consistera à lâcher les bots dans le monde de Second Life à la recherche de leurs objectifs. Dès que le laps de temps défini est écoulé, on stop les bots et on fait une comparaison de leurs données collectées. Si les bots ont tous obtenu des informations proches de leur thème et qu'ils ont enrichi leur base de données, on pourra dire que notre "corps" permet bien la généricité des actions.

Attention : la certification de ce test dépend surtout des différences d'interprétations des résultats, il nous est donc pas rééllement possible de valider ces tests de façon rigoureuse.

4.1.3 Utilisation d'AIML

4.1.3.1 Description du besoin

Le bot de l'ancien groupe utilisait AIML. Dans un premier temps, nous envisageons d'utiliser une structure AIML, que l'on pourrait gérer de manière dynamique. Par exemple, considérons que le bot sache qu'il existe des chocolats. Imaginons que le bot rencontre un joueur avec lequel il discute de ce sujet, il pourrait alors acquérir de nouvelles informations sur les chocolats (il en existe des noirs, blancs...) et ainsi améliorer ses connaissances (sa structure AIML).

4.1.3.2 Tests de validation

Résumé : Sauvegarder la base de donnée avant utilisation. Laisser le bot récolter des informations. A la fin, on compare sa base de donnée courante avec

la sauvegarde. Si de nouvelles entrées sont présentes, l'objectif est atteint.

Détails : Pour tester la bonne utilisation d'AIML et surtout de notre AIML dynamique, nous effectuerons une comparaison entre les fichiers AIML d'origine et ceux obtenus après plusieurs discussions de notre bot avec une copie plus intelligente de lui même qui lui parlera des sujets qu'il connaît mieux que son double. Pour que le test soit considéré comme réussi, il faut absolument un enrichissement de sa base de donnée AIML, le bot "moins intelligent" devra avoir enregistré au moins 30% des informations de l'autre bot. Mais aussi vérifier que le contenu rajouté est pertinent.

4.1.4 Temps de réponse du Bot après analyse de la réponse de l'humain

4.1.4.1 Description du besoin

Ce besoin se découpe en deux parties. Le premier besoin vient du fait qu'il faut rester crédible envers l'humain qui parle à notre bot. En effet, une réponse ne doit pas être instantanée. Un humain tape entre 30 et 60 mots par minute. Si l'humain s'aperçoit que les réponses sont instantanées, il risque de se douter du côté machine de son interlocuteur. Pour se faire il faut établir une stratégie sur le temps de réponse moyen qui dépendra du nombre de mots de la réponse. Selon le type de bot que l'on utilisera, ce temps de réponse pourra varier. En effet, si notre bot se fait passer pour un enfant, son temps de réponse sera plus long que pour un joueur confirmé. La deuxième partie de ce besoin vient de la complexité de l'analyse de la question ou réponse de l'interlocuteur. Selon les cas, notre bot utilisera sa base AIML ou s'il est dans une impasse, utilisera une requête via l'**API google** pour améliorer sa réponse. Ce temps d'analyse est à enlever du temps moyen parlé dans le paragraphe précédent. Un prototype d'une telle requête nous permettra de mieux situer le temps nécessaire.

4.1.4.2 Tests de validation

Résumé : Interroger le chatbot sur des mots qui lui sont inconnus. Si le temps de réponse est toujours inférieur à 15 sec alors le besoin sera considéré comme satisfait.

Détails : Pour tester si le temps de réponse est correct, il faudra tester notre bot dans des domaines qu'il ne connaît pas. En le mettant en difficulté, nous verrons le temps qu'il met en recherche. Si ce temps est estimé trop long (15 secondes), on pourra répondre à l'humain une phrase type, du genre "excuse me, my phone rings, I'll be back in few minutes ?". Tout en veillant à ce que ce genre de réponse types ne se répète pas.

4.1.5 Le bot doit être anglophone

4.1.5.1 Description du besoin

L'anglais étant la langue la plus couramment utilisée sur Second Life, nous devons donc faire de notre bot un anglophone afin qu'il ait un champs d'action plus important. En effet, celui-ci cherchant à "s'intégrer" dans la communauté de joueurs, il se doit de pouvoir communiquer avec la majorité d'entre eux. Nous mettrons au point une stratégie d'insertion d'erreurs de grammaire et d'orthographe afin d'être plus crédible dans le dialogue. De plus, pour éviter de reproduire dans les réponses du bot des erreurs commises par l'interlocuteur humain (ce qui ne serait pas très discret, voir paragraphe "Paraître humain"), nous devrions être à même de détecter ces erreurs.

4.1.5.2 Tests de validation

Résumé :

- Dans le cas où la langue utilisée par l'interlocuteur est inconnue du robot (tous sauf l'anglais), ce dernier devra être capable de le signifier à son interlocuteur ;
- Sur une journée de recherche (24h complètes), le bot devra utiliser plus de 500 mots différents ;

Détails : Pour vérifier que ce besoin est bien pris en compte par le bot, il nous suffirait d'aller à sa rencontre en relevant ses coordonnées dans le jeu, puis de discuter avec lui en anglais afin de connaître sa capacité de communication dans cette langue. Nous tenterons de communiquer avec lui dans une autre langue de façon à savoir comment il réagirait devant un joueur ne sachant pas communiquer en anglais.

Pour pousser un peu plus loin ce test, on peut laisser notre bot discuter toute une journée et étudier combien de mots, il a utilisé. D'après des sites spécialisés [1] Une personne parlant anglais utilise plus de 10000 mots. Il faudrait alors voir combien notre bot utilise de mots. Bien sûr il n'utilisera pas autant de mots, mais si le nombre de mots utilisés est conséquent (> 500 mots), on estimera que notre bot à une bonne connaissance de la langue anglaise (suffisante pour Second Life). Le nombre de mots sera compté de façon automatique, les pluriels seront considérés comme les mêmes mots que les singuliers (dot et dots compterons donc pour un seul mot).

4.1.5.3 Risques liés

Il existe deux risques quand à cette partie :

- notre anglais n'est pas parfait, notre bot aura donc un "parler" à l'image de notre niveau. Cependant, ce problème est mineur car il y a beaucoup de joueurs qui ne parlent pas dans leur langue natale sur Second Life,

1. Site spécialisés :
   - `http://www.peace.co.uk/1/1.html` ;
   - `http://englishenglish.com/english_facts_12.htm` ;

même ceux qui sont anglophones ne parlent pas forcément un anglais correct (sms, 1337=Leet. . .) ;
— détecter que la langue parlée par l'utilisateur est ou non de l'anglais est un problème difficile dans les "conditions Second Life " à cause du "mal parlé" ;

4.1.6 Intégrer à son discours des fautes d'orthographe, frappe de façon aléatoire

4.1.6.1 Description du besoin

Pour être crédible au sein de la communauté de Second Life, nous avons compris que notre bot devait commettre des erreurs d'orthographe ou de grammaire. Mais il faut tout de même que notre bot reste compréhensible (nous ne souhaitons pas utiliser le langage dit "texto"). Pour cela, le bot devra être en mesure de produire des erreurs avec une régularité réglable par l'utilisateur (en fonction de ses besoins) de façon relativement simple (fichier de configuration par exemple). Pour augmenter ou diminuer les fautes volontaires, nous établirons une échelle de valeur (5 étant le plus d'erreurs et inversement, 1 très peu de fautes). Ces erreurs pourront être des fautes d'orthographe, de grammaire ou de frappe.

4.1.6.2 Tests de validation

Résumé : Lancer le chatbot sur Second Life avec divers niveaux de taux de fautes. Discuter avec lui. Conclure sur le nombre de fautes présentes dans ses phrases.

Détails : Afin de tester le fait que les erreurs volontaires ne soient pas néfastes à la compréhension, il nous faudra tester notre configuration de taux de fautes de manière ascendante, en commençant par la valeur de fautes la plus faible. Si cette valeur correspond à une bonne compréhension, on passe à la valeur suivante. En discutant avec notre bot, nous pourrons utiliser ces différentes valeurs de réglage et tester rapidement, si notre bot reste compréhensible.

4.1.6.3 Risques liés

Les risques ici sont simples, si la partie correction d'erreurs ne fonctionne pas correctement, alors, ici nous intégrerons des erreurs en plus de celle reproduite (fautes d'orthographe non corrigées par exemple). Cela pourrait rendre certaine de nos phrases complètement incompréhensibles. Les risques liés à la correction d'orthographe étant assez grand, ce risque est donc quasiment aussi présent. Cependant, encore une fois, ce risque est négligeable car sur Second Life le bon parlé ne court pas les rues.

4.2 Les besoins fonctionnels

4.2.1 Méthode de recherche d'informations (ou sont les informations)

4.2.1.1 Description du besoin

Le bot devra être en mesure de trouver des lieux intéressants pour sa recherche. Il interrogera dans ce but toutes les personnes qu'il croise, si la personne interrogée lui dit qu'elle possède ces informations alors le bot devra poursuivre la discussion. Pour cette recherche, on utilisera comme base un **formulaire générique** à la manière des sondeurs téléphoniques en adaptant les questions en fonction des mots clé du thème choisi.

Exemple de conversation que le bot pourrait avoir, il commencerait avec une phrase d'accroche sur le thème choisi par l'utilisateur : "Do you know someone can explain me *this* ?" **ou bien** "Where can i get informations about *this* ?" , ensuite, si son interlocuteur lui dit qu'il sait des choses à ce propos, il pourrait continuer de l'interroger : "Can you talk me about *this*"

Enfin, si l'interlocuteur lui dit qu'il sait où/au près de qui il pourrait trouver ce genre d'informations, alors le bot pourra également lui demander si lui ne possède pas quelques informations intéressantes également : "You don't know anything about *thi* ?" Si le joueur lui répond que si, alors le bot pourra continuer la conversation **et** stocker le lieu "intéressant" pour y aller ensuite.

Ainsi, on pourrait exploiter la réponse pour trouver les informations de façon plus efficace. Pour finir, on peut orienter la conversation seulement sur la recherche d'informations, donc si l'interlocuteur commence à poser quelques questions, le bot adoptera une stratégie plus autoritaire : "I prefer when I ask questions ! !" ou bien encore plus niaise : "I don't know, I only get information about *this*".

4.2.1.2 Tests de validation

Résumé : Discuter avec le bot et :

- vérifier qu'il nous interroge sur le sujet désiré après au plus 2 réponses de notre part ;
- lui donner des informations sur des lieux, puis vérifier qu'il les stockes correctement ;

Détails : Après avoir engagé la conversation, nous attendons que le Bot nous pose la question. Il faut qu'il la pose vite, sinon le test serait considéré comme raté car un personnage humain ne serait sûrement pas suffisamment patient (au deuxième échange maximum).Après l'avoir renseigné ou au contraire, ne l'avoir rien dit (tout le monde ne sait pas tout), on vérifiera comment l'information est stockée.

4.2.2 Trouver les personnes à interroger

4.2.2.1 Description du besoin

Le Bot devra pouvoir repérer qu'il y a des personnes dans son entourage proche (géographiquement parlant) et ainsi engager une conversation dès que possible (on ne veut pas d'un Bot timide). Le bot pourra "crier sur la voie publique" des phrases d'accroches ou entamer une conversation sur le thème avec "le premier venu". Le bot devra être capable de choisir seul la personne à interroger selon ses critères. S'il arrive dans un endroit avec une forte densité de joueurs, il devra choisir parmi toutes ces personnes, laquelle interroger. On peut envisager de stocker le nom des avatars qu'il connaît déjà et/ou établir une liste de gens qu'il a déjà côtoyé avec un **indice de satisfaction**. Une personne lui ayant déjà donné une bonne information sera susceptible de lui en redonner une. Le bot devra interroger en priorité les avatars qui lui sont inconnus. Il faut envisager aussi de réinterroger les avatars à haut taux de satisfaction régulièrement en intégrant une variable temps aux avatars connus.

Prévoir le cas où le bot se retrouve dans une zone non ou très peu peuplée, dans ce cas il devrait être capable de se déplacer seul afin de rejoindre une zone où la population est plus dense. Cela peut aussi jouer sur le degré d'humanisation du bot (sur la forme du déplacement, s'il tient compte de son environnement ou s'il avance, de façon arbitraire, et dès qu'il rencontre un obstacle, change sa direction de façon à contourner le problème).

4.2.2.2 Tests de validation

Résumé :

- On place le bot dans le monde second life, si lorsque d'autres avatars sont proche il engage la conversation, alors l'objectif sera considéré comme atteint.
- Isoler le bot dans Second Life , se positionner non loin de lui. S'il parvient à établir un contact avec nous, lui fournir des informations, partir puis revenir ensuite. Se souvient-il de nous ? Si oui, l'objectif est atteint.

Détails : Il suffit de placer le bot dans la rue, immobile, voyons s'il engage la conversation lorsque nous passons à coté de lui (avec un second compte). Il faudra vérifier que les personnes stockées dans son historique sont réinterrogées de manière régulière, si elles sont présentes bien évidemment. Pour ce faire, on stocke d'un côté l'historique des personnes déjà interrogées et d'un autre côté, les personnes croisées. En faisant le rapprochement des deux, on verra si notre bot exécute correctement son travail.

Il nous faudrait aussi tester que celui-ci se déplace de façon autonome s'il n'a personne à proximité. Pour cela, nous le placerons dans une zone vide et nous l'observerons de loin ou en caméra aérienne.

4.2.3 Conserver des historiques de conversation synthétisés

4.2.3.1 Description du besoin

Conserver les conversations que le bot considérerait comme pertinente (dans sa recherche de bots). Si le bot pense qu'une conversation à un intérêt, il pourrait alors garder des traces de cette conversation. On pourrait alors vérifier si nos algorithmes de détection sont efficaces. Nous mettrons également des indices de pertinence sur les conversations. Selon le type de réponse (mots clés), le bot établit un indice de pertinence qui déterminerait le degré de pertinence des informations récoltées lors de cette conversation.

Enfin, il est aussi intéressant de conserver les conversations que le bot considère comme non pertinente, le bot n'utilisera pas cette base de conversation pour sa recherche mais nous (en tant que développeurs) pourrions utiliser cette base pour vérifier que le bot sait identifier correctement l'intérêt d'une information. Cet historique pourrait néanmoins être utilisé par le bot pour qu'il paraisse plus humain (ne pas avoir 2 fois la même conversation avec un joueur).

Ces différentes conversations seront stockées et triées selon leur pertinence afin de simplifier notre travail consistant à vérifier que le bot saura considérer une donnée comme intéressante ou non.

De plus, nous devons mettre en place un système de synthèses des informations. En effet, afin que nous ne passions pas des jours entiers à relire les historiques de ses conversations avec les joueurs/bots, celui-ci devra faire un résumé des informations qu'il a récoltées. Par exemple, le bot devra être en mesure de fournir un échantillon de quelques lignes (20-30) des conversation qu'il aura eut pendant une nuit de recherche (10 heures). De plus, nous synthétiserons sous une seconde forme les informations récupérées par le bot. En utilisant les mots clés trouvés lors des conversations, nous mettrons en évidence (gras) ces derniers afin de permettre une lecture en diagonale plus aisée et tout aussi compréhensible que le dialogue entier. Le fait d'avoir deux formes de rendu pourrait être intéressant car tout les utilisateurs n'ont pas les mêmes attentes sur le rendu, notamment quand à sa précision sur les informations récoltées.

4.2.3.2 Tests de validation

Résumé :

- Communication avec nous sur un thème fixé :
  - Le bot possède déjà une définition du sujet énoncé. Dans ce cas, il doit recouper les informations afin de différencier le vrai (réponse la plus courante) du faux ;
  - Le bot ne possède pas de définition du sujet énoncé. Il doit alors récupérer les mots-clés de la réponse reçue, puis utiliser des requêtes à Google pour vérifier la pertinence des informations perçues ;
- Communication avec les avatars présents dans Second Life :

La discution étant libre, l'avatar devrait sans cesse aller chercher des informations ailleurs que dans sa base de donnée. Le but étant qu'il arrive à trouver les informations pertinentes par rapport à son sujet d'étude. Le stockage de ces données doit être pris en compte.

Détails : Pour tester la bonne analyse du bot en fonctions des données que son interlocuteur lui divulgue, nous partirons sur trois bases différentes. Dans les deux premiers cas, le bot sera en communication avec nous afin que nous puissions le tester avec des mots clés choisis plus ou moins pertinents selon notre bon vouloir (les mots clés sont des adjectifs ou compléments circonstanciels et les mots inconnus jusqu'a présent). Par exemple, nous établirons comme sujet de conversation le thème suivant : un éléphant. Le bot nous posera d'abord la question suivante :

— "What is an elephant ?"
— "Elephants are land mammals of the order Proboscidea and the family Elephantidae. There are realy huge and heavy and most of the time, they are grey ! They leaved in the savane but you can see some of them in circus."

Cette conversation nous paraît assez pertinente car elle mêle des informations nécessaires comme ses caractéristiques, des informations pas tout à fait correcte (couleur qui peut varier), une énumération d'adjectifs puis une information étrange qui va lier son lieu d'habitat et un lieu possible de rencontre (un cirque. . .) ;

Deux choses :

— Soit le bot possède déjà une définition plus ou moins précise de ce thème. Dans ce cas, il faudra qu'il soit capable de trouver les mots clés dans l'énoncé de l'interlocuteur et de recouper ces informations avec celles qu'il a en mémoire. En cas de non exactitude, soit il se réfèrera à ses statistiques (si 10 personnes lui ont donné une définition similaire et que l'on est les seuls à donner une autre, il ne prendra pas en compte cette dernière) où si celles si sont nulles ou équivalente à 50% d'exactitude, le bot devra pouvoir effectuer une requête à Google en utilisant les mots clés donnés par l'interlocuteur et analyser les réponses retournées.
— Soit le bot n'a aucune définition en mémoire et dans ce cas, il doit analyser la réponse de l'interlocuteur afin d'en retirer des mots-clés. Une fois cette étape effectuée, le bot doit recueillir des informations grâce à une API Google permettant d'effectuer des requêtes sur le moteur de recherche de même nom. Le bot devra faire une comparaison entre ce que lui a signaler l'interlocuteur (les mots clés) et les résultats obtenus grâce à l'API. En comparant ces divers résultats aux 10 premières pages référencées par Google, nous pouvons nous lancer dans une analyse comparative de certains mots clés entre le contenu de ces pages web et le contenu de notre conversation :

Enfin,

— si le rapport entre les mots clés de ces pages web et ceux de notre conversation est supérieur 0.6 (au moins 60% des mots clé des pages web sélectionnées se trouvent dans la conversation) on considère que cette dernière est pertinente ;
— sinon on ne la considère pas comme pertinente mais nous la garderons pour que nous puissions contrôler par nous même que la conversation est réellement non pertinente ;

Lorsque tout ces tests auront été effectués, le bot pourra sera alors testé avec de vrais joueurs afin d'avoir des conversations plus imprévisible. Ce test se

déroulera exactement comme ceux nous impliquant dans le dialogue, sauf que un réel interlocuteur jouera notre rôle.

Si à la lecture la synthèse un être humain lambda y trouve du sens et une certaine cohérence, nous pourrons considérer que le test est validé.

4.2.4 Histoire et personnalité du bot

4.2.4.1 Description du besoin

Afin de réussir au mieux sa mission, notre bot se doit d'être crédible dans son rôle. Pour se faire, il faut qu'il ait son histoire propre et ses propres caractéristiques pour ressembler à un humain. Les agissements d'un humain sont dictés par son caractère, son facteur social et environnemental. Notre bot se doit d'essayer de ressembler au comportement humain. Après avoir analysé le projet effectué par l'ancien groupe de PDP sur ce sujet, nous avons pu observer qu'ils avaient mis au point un fichier regroupant une grosse quantité d'informations créées pour personnaliser le bot. Ces caractéristiques pourront être personnalisées selon le type de renseignements recherchés. Voici une liste non exhaustive de ces caractéristiques :

— nom ;
— prénom ;
— âge ;
— ville RL (Real Life) ou SL (Second Life) ;
— son histoire (métier, faits marquants de sa vie... ;
— traits de caractères principaux influant sur les questions et l'attitude du bot dans sa manière de parler ;
— traits physiques spécifiques (ex : petit gros, bimbo...) ;

Lors de conversations avec des avatars humains, le bot utiliserait son histoire si l'humain lui pose des questions spécifiques. Cela augmenterait la crédibilité du bot mais aussi la confiance que l'humain lui donne. Si le bot arrive à gagner la confiance de son interlocuteur, il glanera plus facilement des informations.

4.2.4.2 Tests de validation

Résumé : Discuter avec le chatbot en lui posant des questions sur sa vie personnelle, ses goûts, ses loisirs ou son passé. Si celui-ci est capable de récupérer ses données personnelles concernant ces sujets, le test sera considéré comme réussit.

Détails : Nous savons maintenant que le bot devra avoir un passé et/ou un avenir ! Mais afin de vérifier que celui-ci soit au courant du fait qu'il a existé, nous pourrions tout simplement établir une communication avec lui comme par exemple :

— Favorite animal ;
— Birthday ;

Puis, si celui-ci arrive à réagir à des mots-clés comme ceux-ci, nous tenterons alors une communication plus avancée :

— What is your favorite animal ?

— What is your birthday ?
— Do you love music ?
— What is your book bedside ?
— In which city do you live in the real world ?

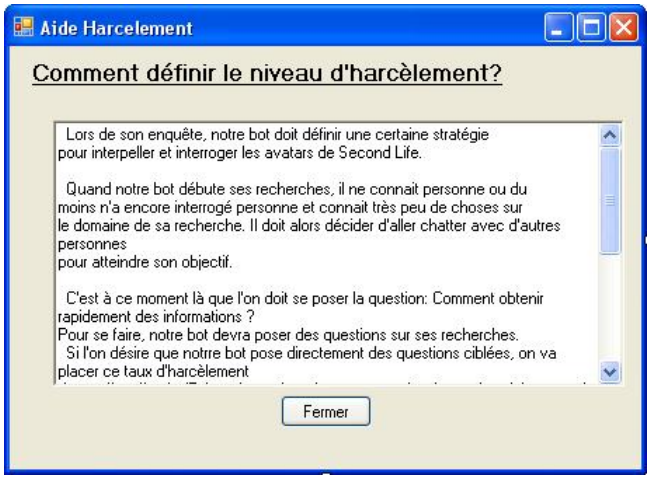
Si le robot est capable d'aller chercher dans ses informations personnelles de telles informations et de nous les transmettres, nous pourrons alors affirmer que notre robot est capable de simuler son existence, d'avoir une mémoire et donc de se rapprocher des humains.

4.2.5 Aider l'utilisateur à éditer un fichier de "configuration"

4.2.5.1 Description du besoin

Pour définir les critères de recherche du bot, il faudra éditer un fichier (AIML) pour définir son objectif (quelles questions posées). En effet, ce fichier à une structure qui n'est pas forcément connue de tous, c'est pourquoi il nous parait nécessaire de fournir une aide à la saisie d'un tel fichier. Nous proposerons par exemple une interface toute simple qui permettrait de demander à l'utilisateur une liste de question qu'il désire poser et également des réponses qu'il souhaiterait donner à des questions que les joueurs poseront au bot. Cet éditeur devra également permettre de définir le niveau de "connaissance apparente de la langue" du bot (voir paragraphe "Langue parlée par le bot"). Voici un aperçu de ce à quoi ressemblera cet éditeur, ici la première page qui permettra de générer le fichier de base :

Maquette : génération de bot



Maquette : Aide

4.2.5.2 Tests de validation

Résumé : Laisser des utilisateurs non-initiés et non impliqués dans le projet utiliser l'éditeur. Chaque utilisateur répondra à une quantité définies (environ 10) questions notées. Si l'application obtient la moyenne sur la totalité des notes données par l'utilisateur, l'éditeur sera considéré comme fonctionnel et utilisable.

Détails : Pour valider cet éditeur, nous proposerons à une partie des élèves de la promotion et également des personnes non informaticiennes de tester l'éditeur, puis de répondre à un questionnaire concernant sa simplicité et son efficacité. Le questionnaire comprendra notamment les questions suivantes :

- L'interface est-elle claire au premier abord ? (exemple de propositions : (5) Tout a fait, (4) plutôt, (3) j'ai mis un peu de temps mais ça reste correct, (2) pffff.... c'était dur, (1) vraiment...je trouve que c'est nul, (0) je ne veux même pas en parler)
- Est-il rapide d'éditer beaucoup de champs ?
- L'éditeur de questions (bot) est-il bon ?
- L'éditeur de correspondances entre les questions (autre joueurs) et réponses (bot) est bonne-il correct ?
- Ai-je réussi à faire tout ce que je voulais ?

Chaque question sera notée sur 5, si la moyenne obtenu sur la globalité du panel est supérieure 2.5 alors l'éditeur répondra correctement au besoins.

4.2.6 Établir des statistiques concernant les réponses données par les joueurs et bots

4.2.6.1 Description du besoin

Afin de mener à bien ses objectifs ou d'améliorer ses connaissances sur le monde de Second Life, le bot devra pouvoir effectuer des statistiques en fonction des données réceptionnées depuis Second Life. Ces données peuvent être de diverses formes. Par exemple, celui-ci pourra souhaiter calculer le rapport entre le nombre de joueurs féminins et masculins dans le jeu, ou encore tirer parti des conversations établies avec les joueurs afin de savoir dans quels lieux le bot aura plus de chance de trouver d'autres robots. Autre chose pouvant nous être utile, il s'agira de repérer une conversation intéressante afin de savoir avec quel joueur mais aussi à quel endroit elle a eu lieu. La première donnée pourra nous servir lors des prochaines rencontres avec ce joueur de manière à savoir si celui-ci peut nous fournir des informations véridiques et/ou pertinentes tandis que la seconde donnée nous permettra de répertorier les lieux fréquentés et où nous pourrions éventuellement trouver de nouvelles informations. Il faut bien sûr avoir en tête que le facteur chance joue un rôle important dans les relations avec les joueurs et donc dans les discussions que le bot aura avec eux.

4.2.6.2 Tests de validation

Résumé :

- Pour les données chiffrées : calculer des statistiques par rapport à un sujet souhaité. Comparer le résultat avec les statistiques officielles. On

pourra accepter une erreur de 10%.
— Pour les données non chiffrées : récupérer les données stockées par le robot. Aller sur Second Life avec un autre compte pour vérifier ces informations. Si elles s'avèrent la plupart du temps exact, le test est réussit.

Détails : Les tests de validation de ce besoin peuvent être séparés en tests qualitatifs et quantitatifs. Si le but du bot est d'établir un sondage, un recensement ou tout autre information pouvant être calculée, on peut imaginer une comparaison avec des études déjà existantes sur Internet (exemple sur le `http://secondlife.com/whatis/economy_stats.php`). Cette partie du site met à jour quotidiennement différentes informations sur la vie de Second Life. Ainsi on pourrait tester notre bot en lui fixant comme objectif de calculer lui même ce genre d'informations, si on arrive à un écart minime (moins de 10%), on pourra dire que le test est réussi.

Concernant les informations récoltées par notre bot qui sont plus générales (lieux susceptibles d'accueillir un joueur possédant une information intéressante, endroits fréquentés par un certain type de population,etc. . .), on devra se connecter avec un compte joueur et vérifier par nous même que les informations vérifiées sont correctes. Le bot aura fait le travail d'investigateur, notre avatar n'ayant plus qu'à aller à l'endroit spécifié et récolter l'information recherchée. Si l'endroit est bien la zone que l'on cherchait, le test sera considéré comme réussi.

4.2.7 Faire le lien entre des points clés et des coordonnées dans le jeu

4.2.7.1 Description du besoin

Utile pour le besoin précédent, si quelqu'un nous indique un lieu, il nous dira très certainement quelque chose du genre "Go to the Moon restaurant at the end of this street". Le bot devra alors se rendre au restaurant cité. Cette réponse devra nous servir pour la recherche, il serait donc important de traduire l'emplacement désigné avec des mots en une coordonnée spatiale.

4.2.7.2 Tests de validation

Résumé : Indiquer une liste de 10 lieux au robot puis observer ses déplacements. Si il passe par 8/10 lieux alors le test sera valide.

Détails : Nous testerons cette fonctionnalité en envoyant le bot à la recherche d'une suite de lieux bien précis que nous aurions repérés (noms et coordonnées spatiales). Son objectif serait alors simplement de se rendre aux points fixés de façon autonome. Le bot devra passer ainsi par 10 points de contrôle. Si le bot arrive au dernier lieux en ayant passé au moins 8 des points alors le besoin sera considéré comme couvert.

4.2.8 Système de correction d'erreurs (orthographe)

4.2.8.1 Description du besoin, méthode

Nous devons corriger les fautes commises par les joueurs humains pour éviter de les reproduires. Nous utiliserons pour la correction d'orthographe les outils de pilotage mis à disposition par Microsoft : *Interop.Word.dll* (outils permettant la création de documents Office en .NET). Ceci nous permettra de trouver des erreurs ; ensuite, il suffira de :

— Corriger directement la faute s'il n'y a qu'une suggestion émise par le correcteur de Word (contenu dans *Interop.Word.dll*)
— Sinon, le bot devra choisir la première proposition du correcteur qui est celle qu'il estime la plus probable ce qui impliquera forcément des erreurs. Cependant, nous ne pensons pas qu'il soit objectivement possible de faire mieux.

4.2.8.2 Tests de validation

Résumé : Discuter avec le chatbot en incluant des fautes d'orthographes dans nos messages. Analyser et comparer la réponse de chatbot avec notre message. En déduire de la bonne fonctionnalité de notre chatbot.

Détails : Pour tester cette fonctionnalité, nous devrons simplement parler avec le bot, en le configurant de manière à ce qu'il ne fasse que répéter nos phrases avec les corrections qu'il aura apporté, ainsi, nous pourrions voir s'il les corriges de façon efficace.

4.2.8.3 Les risques liés à la correction orthographique

Nous devons tout de même être méfiant lorsque nous pratiquons la correction orthographique. Il se peut parfois que notre correction n'apporte qu'un non-sens par rapport à la phrase souhaitée au départ. Par exemple, si l'interlocuteur du bot dit : "Ya plu d'humain la ba"

Nous pourrions le corriger en :
— "Il n'y a plus d'humains là-bas" ;
— "Il y a plus d'humains là-bas" ;
Nous avons choisi ici un exemple en français pour illustrer le problème mais il est évident que le même genre de confusion est possibles en anglais.

Cette incapacité à savoir décider de quelle solution adopter a des chances de nous arriver. Ne serais-ce que sur ce point, la correction orthographique peut révéler des risques pour l'établissement de nos statistiques ou même pour la réussite du bot dans son objectif. Il est également possible qu'une correction ne soit pas possible dans certains cas précis.

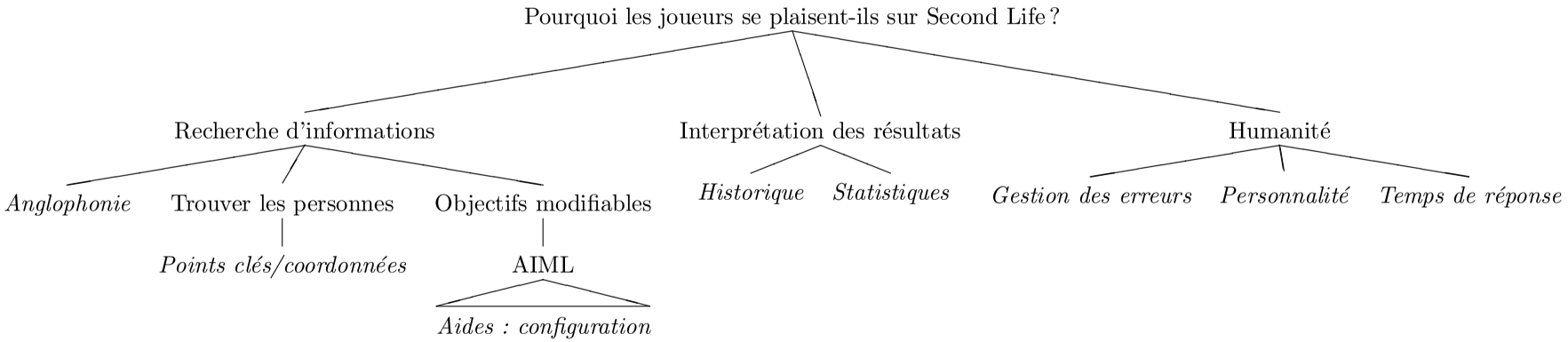
Une idée pour palier à ce problème, le bot pourrait avoir des phrases du type : "I don't understand ..." ou bien "Please tell me the meaning of this sentence", etc...

4.3 Les anti-contraintes

4.3.1 On ne demande pas au bot de passer le Test de Turing avec succès

Le but de ce bot n'étant pas de simuler réellement un comportement humain, nous ne nous efforcerons pas d'obtenir un bot spécialement doué dans ce domaine. Nous devons avant tout rester concentrer sur notre objectif primordial qui est la recherche d'informations. De toute façon, nous savons qu'actuellement aucun chatbot ne passe le Test de Turing avec succès et nous n'avons certainement pas la prétention de dire que nous réussirons en 3 mois ce que des experts n'ont pas réussit en plusieurs années... Simplement, nous devons tout de même mettre au point notre bot afin qu'il puisse se confondre au mieux avec la population humaine de Second Life (voir le paragraphe "Paraître humain"). Afin de tester un minimum l'aptitude de notre robot à détecter si son interlocuteur est humain ou robot, nous pourrions effectuer le test suivant : Si deux bots lancés en parallèle parviennent à se reconnaître mutuellement alors on considère que le test est un succès. En répétant cette opération un grand nombre de fois, on pourrait alors établir l'aptitude du bot à reconnaître un robot grâce à une conversation. De même nous devrons vérifier que le bot ne prend pas des joueurs humains (répondant de façon intelligente) pour des bots. Nous pourrons alors avec un deuxième compte (avec lequel nous nous connecterions à Second Life via l'interface officielle) engager une conversation avec notre Bot pour voir s'il nous prend pour un bot ou non. Il serait intelligent de faire faire ce test à une personne extérieure au groupe de développement car celle-ci ne connaîtrait pas les "faille de détection de Bot" et ne serait donc pas influencée dans son discours.

4.4 Schéma fonctionnel



Remarque : Les fils sont les conditions nécessaires à un fonctionnement optimal du père, cependant, certaines ne sont que des améliorations possibles (et donc pas indispensables). Pour chaque niveau, la fonction la plus à gauche est la plus importante.

Chapitre 5

Langage et bibliothèques utilisés

5.1 Choix du langage de programation

D'après les besoins déjà établis, nous avons obté comme le demandait au départ le client, pour un dévellopement en C#. Voici les raisons de ce choix :

5.1.1 Choix du C#

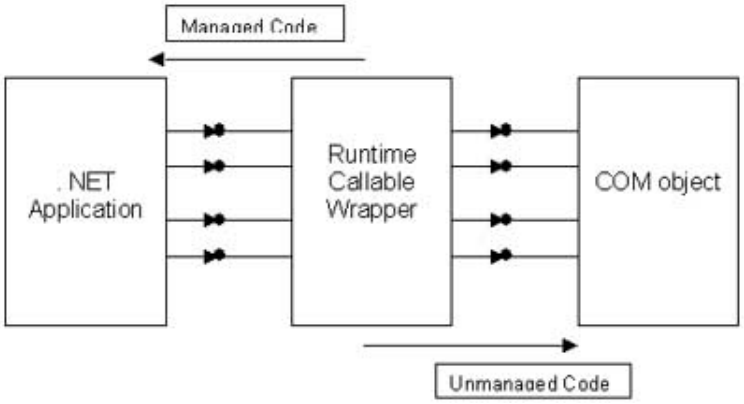
Du fait de l'interopérabilité du langage C#, nous pouvons utiliser d'autres langages que C# qui est le langage avec lequel a été faite la libSL.

> « L'interopérabilité des langages est une fonctionnalité clé du .NET Framework. Comme le code du langage intermédiaire produit par le compilateur C# respecte la spécification de type commun (CTS), lorsqu'il est généré à partir de C#, il peut interagir avec le code généré à partir des versions .NET de Visual Basic, Visual C++, Visual J# ou d'un des vingt autres langages respectant la norme CTS. Un même assembly peut contenir plusieurs modules écrits dans différents langages .NET, et les types peuvent se référencer l'un l'autre exactement comme s'ils avaient été écrits dans le même langage. »
>
> *Microsoft* [1]

Le concept d'interopérabilité est interessant dans la mesure où il propose d'utiliser des bibliothèques, écrites dans un certain langage, avec un autre langage. Etant donné que C# fait partie du Framework .NET, il est possible d'utiliser une bibliothèque C# avec d'autres langages inclus dans ce framework (VB.NET, J#, C++, ...) en respectant certains cirtères. L'interopérabilité avec les autres langages est aussi assurée si ils respectent le standard COM [2] mais

1. `http://msdn.microsoft.com/fr-fr/library/z1zx9t92.aspx`
2. `http://www.microsoft.com/com/default.mspx`

demande plus de ressources car le lien entre le framework .NET et les COM object se fait par le biais d'un composant appelé "Runtime Callable Wrapper".



Source : Csharphelp[3]

Donc bien qu'il soit possible d'utiliser d'autres langages de programmation, nous nous contenterons de développer en C# afin de ne pas rajouter un travail de conversion supplémentaire. D'autant plus que C# possède une syntaxe proche de celle de Java, il nous sera donc plus aisé de nous y adapter. Ainsi, nous ne perdrons pas de temps sur l'aspect aprentissage d'une nouvelle syntaxe.

5.2 Bibliothèques utilisées

En plus du framework .NET 3.5, nous utiliserons la LibSL afin de communiquer avec Second Life. A priori, il ne nous est pas nécessaire d'utiliser d'autres bilbiothèques. Si jamais nous serions amené à communiquer avec des bases de données, nous pourrons le faire en utilisant LINQ qui est directement intégré au Framework .NET 3.5, ce qui nous permettrai de faire directement nos appels aux bases dans le code source en C#.

Linden Lab, les créateurs de Second Life, on developpé un langage de script pour Second Life : **LSL** (Linden Scripting Language). Dans le cadre de ce projet, nous ne pourrons pas l'utiliser car, ce langage est pratique pour faire des petits scripts (taille max des fichier 16Ko), mais inadapté à de plus gros projets et surtout les scripts ne s'appliquent pas aux avatars.

> « A script in Second Life is a set of instructions that can be placed inside any primitive object in the world, but not inside an avatar. »
>
> *Linden Lab*[4]

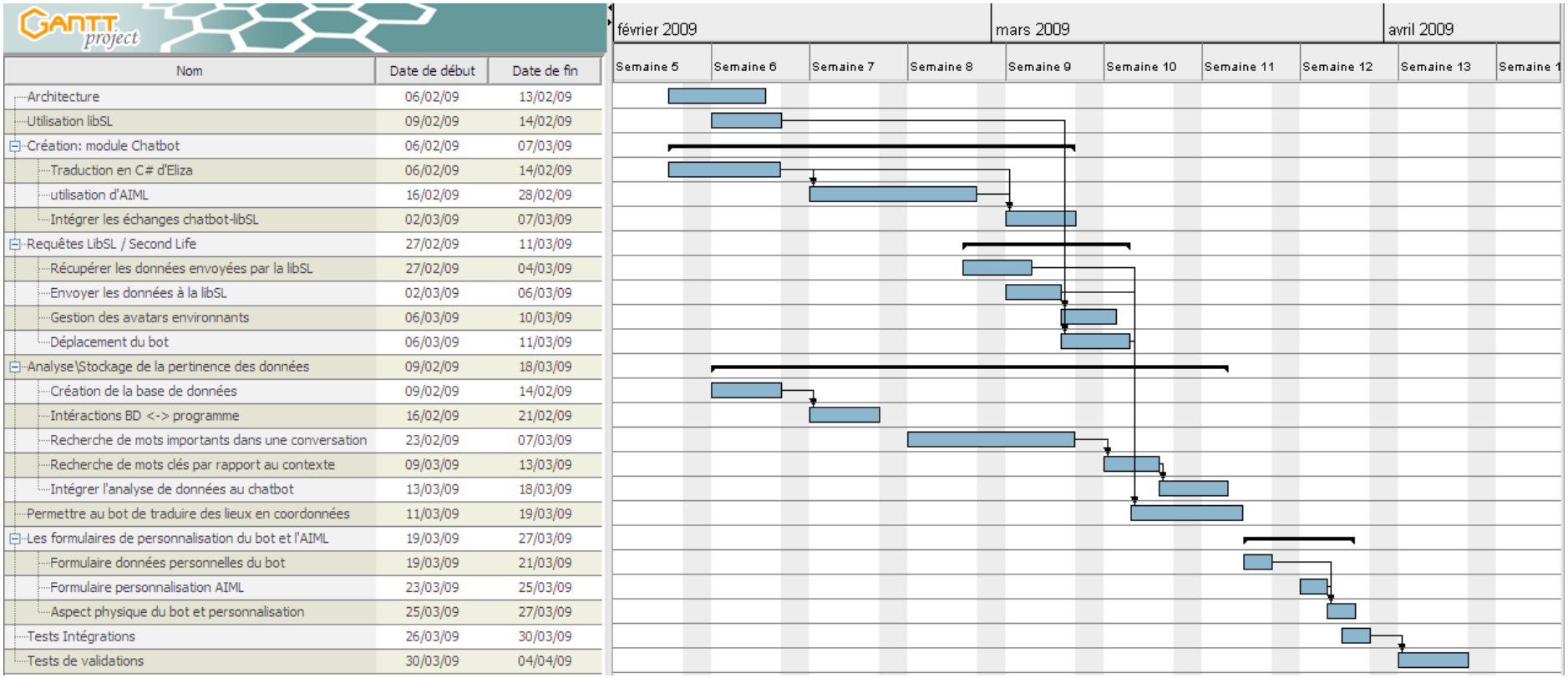
3. Csharphelp : `http://www.csharphelp.com`
4. LSL : `http://wiki.secondlife.com/wiki/Help:Getting_started_with_LSL`

Chapitre 6

Plannings

6.1 Planning prévisionnel

Voici le planning établi le 4 février 2009, disponible dans sa version *GanttProject* en annexes. Sur cette source (projet.gan), vous pourrez voir les ressources associées à chaque tâches. Ci-dessous, un apperçu est disponible



Nom	Date de début	Date de fin
Architecture	06/02/09	13/02/09
Utilisation libSL	09/02/09	14/02/09
Création: module Chatbot	06/02/09	07/03/09
Traduction en C# d'Eliza	06/02/09	14/02/09
utilisation d'AIML	16/02/09	28/02/09
Intégrer les échanges chatbot-libSL	02/03/09	07/03/09
Requêtes LibSL / Second Life	27/02/09	11/03/09
Récupérer les données envoyées par la libSL	27/02/09	04/03/09
Envoyer les données à la libSL	02/03/09	06/03/09
Gestion des avatars environnants	06/03/09	10/03/09
Déplacement du bot	06/03/09	11/03/09
Analyse\Stockage de la pertinence des données	09/02/09	18/03/09
Création de la base de données	09/02/09	14/02/09
Intéractions BD <-> programme	16/02/09	21/02/09
Recherche de mots importants dans une conversation	23/02/09	07/03/09
Recherche de mots clés par rapport au contexte	09/03/09	13/03/09
Intégrer l'analyse de données au chatbot	13/03/09	18/03/09
Permettre au bot de traduire des lieux en coordonnées	11/03/09	19/03/09
Les formulaires de personnalisation du bot et l'AIML	19/03/09	27/03/09
Formulaire données personnelles du bot	19/03/09	21/03/09
Formulaire personnalisation AIML	23/03/09	25/03/09
Aspect physique du bot et personnalisation	25/03/09	27/03/09
Tests Intégrations	26/03/09	30/03/09
Tests de validations	30/03/09	04/04/09

Remarques :
— Toutes les pages web ont été dernièrement consultée le 12 janvier 2009
— Style de la bibliographie : Copyright (c) 2003 Michael Shell (IEEEtran)