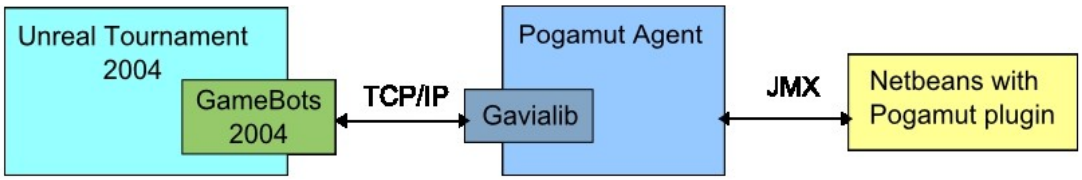
IAS --Labo IA et jeux-vidéo

Ce labo reprend et adapte une partie du Tutoriel de Pogamut

http://pogamut.cuni.cz/pogamut_files/latest/doc/tutorials/TutorialsBook.html

Comportement de bots avec Pogamut et Unreal2

Bot : petit nom pour « robot ». Dans le contexte qui nous occupe ce terme désigne un robot logiciel, un agent virtuel. La bibliothèque Pogamut vous permet de décrire en Java le comportement d'agents qui seront plongés dans le moteur de jeu Unreal 2. Vous pourrez alors les observer avec un client-joueur du jeu.



Ce jeu comporte une partie serveur qui gère tout le niveau et les règles. Les joueurs ainsi que les bots s'y connectent pour incarner des avatars. D'un point de vue architecture, du point de vue du serveur de jeux, les bots que vous allez programmer sont des "joueurs", au meme titre que l'avatar dans le client de jeu que vous allez controller.

Documentation des classes java de pogamut:
http://pogamut.cuni.cz/pogamut_files/latest/doc/javadoc/

Première Partie : lancer un bot, le cycle de vie d'un bot

1. **Creation d'un projet a partir du repertoire Maven de Pogamut, sous Netbeans**

Vous allez créer un projet à partir de sources disponible en ligne à l'aide de Maven.

Lancer Netbeans 7.4

File → New Project

Selectionner **Maven**, puis **Project from Archetype**, **Local archetype Catalog**, **00-empty-bot-archetype**.

Cliquer sur **Next**

Choisir ensuite un nom de projet et terminer la construction.

L'opération, qui peut prendre un certain temps, devrait se terminer par un BUILD-SUCCESS. Faire ensuite un **Clean Built** (menu Run, Clean and Build ou MAJ + F11).

2. **Lancer un serveur de jeu**

Lancer le fichier .bat **startGamebotsDMServer** accessible depuis C:\GOG Games\Unreal Tournament 2004\System

Dans **Netbeans**, cliquer sur l'onglet **Services** (allez dans le menu

Windows si necessaire).

Cliquer-droit sur **UT2004 Servers** et selectionner **Add server**

Dans la boite de dialogues, donner un nom de serveur, puis dans **URI** mettre **localhost** (l'adresse net du serveur). Maintenant, Netbeans connaît le serveur de jeu. Vous devriez pouvoir voir la carte du jeu grâce à l'onglet Services, en double-cliquant sur le nom du serveur créé.

3. **Créer un bot-avatar controlé par un joueur pour interagir ou observer le comportement du bot sur lequel on travaille.**

    - Lancer le client du jeu **Unreal Tournament** accessible depuis le bureau. Connectez vous au serveur.
    - Explorez les contrôles auxquels vous avez accès et les menus disponibles (tapez la touche ESC pour revenir au menu). A quoi correspond le mode **spectate** ?

4. Serveur, Joueur et Bot en meme temps
    - Revenir à Netbeans, et visualisez l'avatar sur la carte du niveau de jeu, dans la fenetre **Services**.
    - Lancez maintenant un bot: dans netbeans, onglet Projects, lancez le projet que vous avez créé à l'étape précédente.

←

4. **Répondre aux questions suivantes :**

Consultez le code des méthodes et précisez leur role :

prepareBot :
getInitializeCommand :
botInitialized :
botFirstSpawn :
beforeFirstLogic :
logic :
botKilled :

Ces méthodes doivent etre redéfinie pour toute classe étendant **UT2004BotModuleController** ainsi que d'autres classes de bases fournissant des comportement génériques pour les bots.

En particulier, naviguez (clic droit) sur le type des paramètres pour voir à quelles informations ces méthodes donnent accès (classes **GameInfo**, **ConfigChange**, **InitedMessage**).

Modifiez maintenant le code :

- Dans quel ordre ces méthodes sont elles appelées ? (Modifiez le code afin de répondre).
- Changez le nom du bot.
- Inspectez plus précisément le code de la méthode **botFirstSpawn** : en utilisant l'aide contextuelle (CTRL + SPACE), et la fenetre Hierarchy de Netbeans (pour explorer les sous-types de **CommandMessage**) regardez quelles autres méthodes sont disponibles pour les objets utilisés. En particulier, examinez la commande **Jump**.
- Modifiez le code afin que le bot saute tous les 10 appels a la méthode

logic.

- Modifiez le code afin que le bot se déconnecte après quelques sauts.

Deuxième Partie : un bot qui réagit aux événements

Au lieu d'utiliser la boucle de comportement autonome définie par la méthode logic, on va ici créer un bot qui se contente de réagir aux stimulus recus du monde extérieur.

1. **Créer un nouveau projet avec l'archetype maven 01-responsive-bot-archetype.**
2. **Lancez le bot.** Lancez un client de jeu (pas en mode spectate). Observez ce qui se passe lorsque votre avatar approche le bot, et lorsque vous le bousculez.
3. **L'interface IworldView** est l'un des premier fichiers importés par le code source java de ce bot. Naviguez (clic-droit) dans le code de cette interface. Elle sert à la fois à représenter les « sens »du bot et de mémoire simplifiée. Elle permet de manipuler des objets divers (joueurs, éléments d'inventaire…) grâce au type **IWorldObject** et des événements (entend un son, se cogne contre un mur…) grâce au type **IWorldEvent**.

Explorez l'ensemble des sous types de **IworldEvent** , et plus particulièrement: quel est l'événement qui correspond à la perception d'un bruit ? A quoi correspond l'événement **Bumped** ?

On peut associer des méthodes à des événements grace à des annotations. Par exemple, cherchez dans le code de l'exemple la méthode **bumped**. Elle est annotée de **@EventListener(eventClass = Bumped.class),** ce qui signifie qu'elle est appelée à chaque fois que l'événement **Bumped** est reçu par le bot.

Quels sont les autres méthodes ainsi annotées ? A quels événements correspondent ils ?

Modifiez le code pour que suite a 5 collisions avec un autre avatar, le bot suive en l'insultant le dernier qui l'a bousculé (s'il peut le voir), voire lui tire dessus.

Trouvez dans le code une autre manière de lier événement et comportement, sans utiliser les annotations. C'est en lien avec le design pattern Observateur.

Troisième partie : un bot qui se déplace en utilisant le graphe de navigation

On va ici utiliser le graphe de navigation sur la carte pour faire se déplacer le bot. Suivant les principes étudiés en cours, la carte est couverte de nœuds (navpoints) reliés par des arretes, les nœuds possédant des informations sur la manière de se deplacer (par exemple, s'il faut sauter pour passer au nœud suivant).

1. **Créer un nouveau projet avec l'archetype maven 01-navigation-bot-archetype.**
2. Lancez le bot. Lancez un client de jeu ou un mode spectate. Observez ce qui se passe. Tapez au clavier (CTRL + G). Suivez le bot quelques instants et décrire ce qui se passe.

3. Explorez le type NavPoint. Citez les informations qui peuvent être enregistrées

dans cette structure.

4. Répondre aux questions suivantes :
   Quel est l'objet responsable de trouver le chemin dans le graphe ?
   Quel est l'objet responsable de faire suivre au bot le chemin trouvé dans le graphe ?
   A quoi sert l'objet **tabooNavPoints** ?
   Que se passe t-il si le bot est coincé quelque part ?

5. Modifiez le bot pour qu'il ne commence à courir que lorsqu'un autre acteur lui rentre dedans.
6. Modifiez le code pour qu'au bout d'une vingtaine de secondes, il essaye de se rendre à un point de départ (point de respawn).

Synthèse des parties 2 et 3:

On a vu en cours qu'une manière traditionnelle de représenter les personnages non joueurs utilise les automates à état. Définissez des classes Java permettant de représenter le comportement du bot selon son état d'esprit, et dont les transitions seront des événements.

Exemple (à élaborer en fonction de votre maitrise de la bibliothèque et du temps qu'il vous reste) :
Lorsque le NPC s'ennuie, il erre au hasard le long du NavMesh.
Lorsqu'il est bousculé, il poursuit le joueur.
Au bout d'un certain temps, il retourne à un point de respawn.

Quatrième partie: Réactive Planning

Posh est un moteur de planification réactive qui permet de définir un ensemble hiérarchisé de buts et la manière de les réaliser. Chaque évaluation de plan décide quel besoin le bot doit satisfaire et de quelle manière.

1- Ouvrir et construire le projet emohawk-sposh-mood-bot
Dans le répertoire « Other Sources » double-cliquer sur le fichier BotPlan.lap. Ceci ouvrira l'éditeur de plan. Vous pouvez consulter l'editeur graphique et le texte de description.

Le plan contient une liste de « drives », ce que le bot voudrait faire. Chaque « drive » est constitué de « trigger » (ce qui les déclenche, des conditions) et d'actions ou séquences d'actions : ce que le bot exécutera alors.

Les « drive » ont des ordres de priorités, ce qui permettra au bot dont les « sens »percevront plus .

2- Examinez le code : comment se comportera ce bot ? Faites tourner le programme pour vérifier votre hypothèse, et utilisez le mode spectateur ainsi que le mode joueur pour voir les différences éventuelles. Echangez, grâce à l'éditeur graphique ou textuel du plan, les actions proposées pour les drives. Vérifiez le comportement de votre bot.

3- Ouvrir et construire le projet emohawk-modular-sposh-mood-bot, et examinez le code.