Laboratorio 2

Desarrollo Basado en Agentes

Grupo 305 WING

UNIVERSIDAD
DE GRANADA

I Lab 2: Capture the lost Jedis

1 Diseño del Agente. Recarga de energía.

En el primer laboratorio implementamos un agente que representa un vehículo aéreo de tipo *Tie Shuttle*.

En este segundo laboratorio, era necesario determinar una heurística para solucionar el problema de recarga de energía.

1.1 Idea inicial

Nuestra idea inicial fue añadir un Choice dentro del Decision Set que estuviera destinado a recargar, y que se llamara **RECHARGE**.

Para implementarlo, modificamos la función `U(Environment E, Choice a)`:

- En caso de estar por encima del suelo y que la batería estuviera debajo de un umbral, llamaba a la función `goLanding()`, que prioriza la acción de descender.
- En caso de estar en el suelo y que la batería estuviera por debajo del umbral, llama a la función `goRecharge()`, una función nueva que se encarga de priorizar la acción de recargar.

Además, también modificamos la función que mueve al agente a la ciudad, antes llamada `MySolveGoal()`, y que de ahora en adelante se llamará `MyMoveCity()`, para que en caso de encontrarse con la acción de **RECHARGE** al ejecutar el plan, proceda a ejecutar la función `AskForRecharge()`, que ejecuta toda la secuencia de comunicación con el agente BB1F destinada a la recarga, tal y como explicaremos a continuación.

Sin embargo, esta idea no funcionó correctamente porque no actualizaba las percepciones e intentaba pedir una recarga sin estar en el suelo.

1.2 Idea final

Para solucionar el problema simplificamos todo lo anterior mediante la inserción de un bucle al principio de la función `MyMoveCity()` para que en caso de que la batería se encuentre por debajo de un umbral, el agente baje hasta estar en el suelo, y después de leer las percepciones, ejecuta la función `AskForRecharge()`.

Para determinar el umbral a partir del cual se considera que es necesario recargar la energía, es necesario tener en cuenta que nuestro agente va siempre a una altura fija por encima del suelo, en este caso 15. Por eso mismo, después de probar distintos valores para el umbral, decidimos mantener el valor final (50). En la siguiente práctica puede ser necesario mejorar esta decisión.

2 Diagramas

Todos los diagramas se encuentran disponibles en el *siguiente enlace*, en caso de que sea necesario visualizarlos de una forma más clara.

2.1 Diagrama de estados

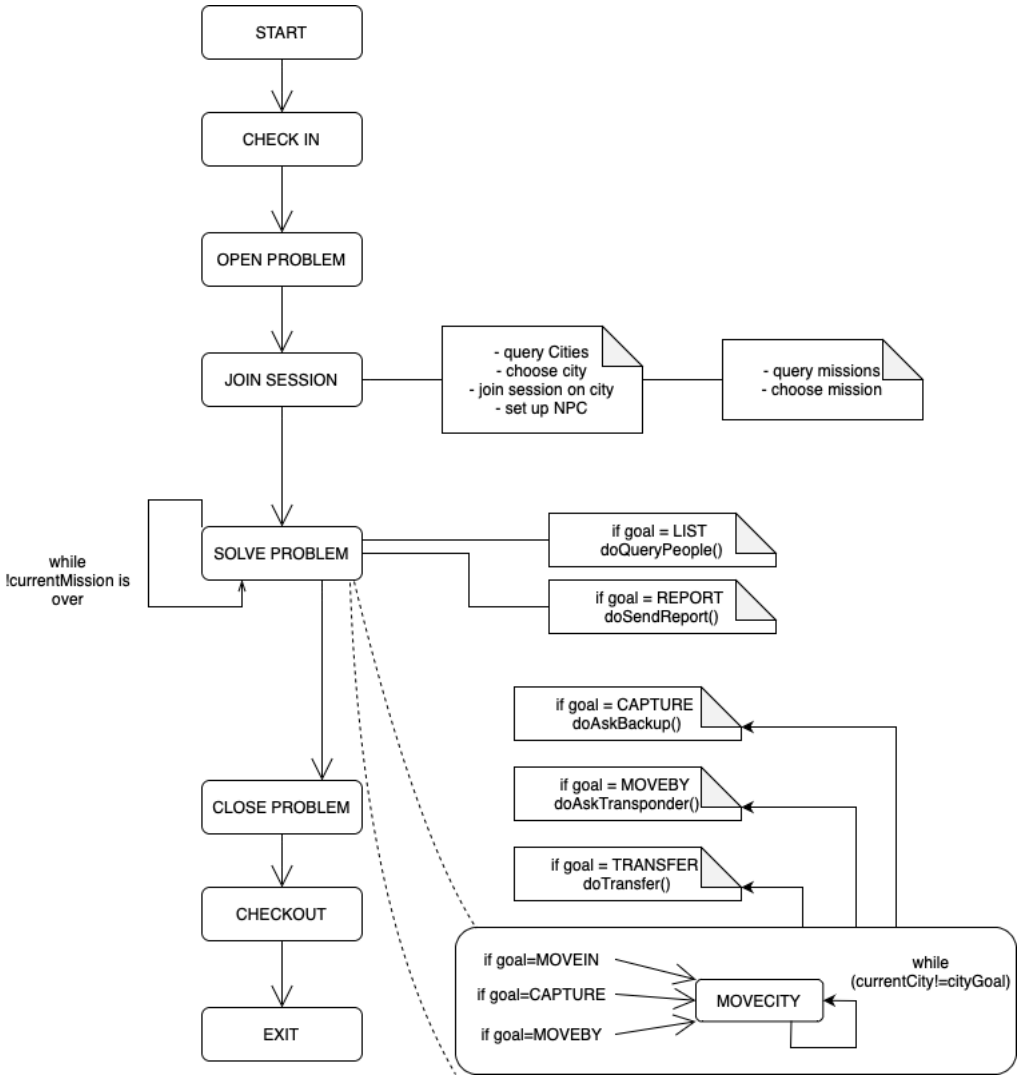

Figura 1: Diagrama de estados del agente TS

Como vemos, hemos mantenido la estructura inicial de estados, realizando cambios en los siguientes:

- **SOLVEPROBLEM**: en este estado iteramos a través de los objetivos de la misión (mientras ésta no esté completada), si el objetivo considerado es de tipo *LIST*, *REPORT* o *TRANSFER* lo solucionamos convenientemente, y si es de tipo *MOVEIN*, *CAPTURE* o *MOVEBY* pasamos al estado **MOVECITY**.

- **MOVECITY** (previamente llamado SOLVEGOAL): en este estado utilizamos el AUTONAV para ir estableciendo rutas a objetivos intermedios. Mientras no se esté en la ciudad objetivo, buscamos una ruta nueva, y tras ejecutar todos los pasos, leemos los sensores, para que la siguiente búsqueda de ruta se haga correctamente. Una vez lleguemos a la ciudad objetivo, volvemos al estado **SOLVEPROBLEM**.

Tal y como hemos visto, el estado **MOVECITY** se encarga de llevar al agente a la ciudad correspondiente. La novedad en esta práctica es que tenemos dos objetivos nuevos que requieren que el agente se mueva a la ciudad: *CAPTURE* o *MOVEBY*.

En *CAPTURE*, es necesario movernos a la ciudad en la que hacemos las capturas, y para ello entramos en el estado **MOVECITY**. Una vez estemos en ella, se comunica con un agente MTT (entraremos en detalle sobre esa comunicación más adelante), y llama a la función `doCapture()`, que se encarga de actualizar el número de personas en la ciudad (`doQueryPeople()` modificado para no actualizar el report), y de mandar los mensajes correspondientes para capturar. Cuando termina, libera al MTT.

El segundo objetivo nuevo, *MOVEBY*, tenemos que obtener la ciudad a la que queremos ir, mediante el transponder de un agente DEST de nuestra sesión. Una vez hemos conseguido el nombre de la ciudad, entramos en **MOVECITY** para llegar hasta ella.

3 Diagrama de secuencia

Hemos decidido no incluir el diagrama de secuencia completo en esta memoria porque complicaba la comprensión de la misma. Sin embargo, en el diagrama podemos observar que una vez entramos en el estado **SOLVEPROBLEM** y llegamos al bloque principal, tenemos nuevas opciones (CAPTURE, MOVEBY y TRANSFER).

También hay que tener en cuenta la modificación de la función `LARVAblockingReceive()`, para incluir una posible solicitud de transponder, en cuyo caso la resuelve y pasa a los siguientes mensjaes.

En la opción `CAPTURE`:

- Utilizamos los mensajes que ya explicamos en la práctica anterior para utilizar el AUTONAV hasta la ciudad correspondiente.
- Ejecutamos la función `doAskBackup()` que se encarga de pedirle al DF un agente de tipo MTT disponible que esté en su sesión, iterando sobre la lista que recibe hasta que uno acepte. Cuando uno acepte, le mandará una solicitud de transponder, y si la respuesta a esa solicitud es correcta, el MTT responderá con AGREE, y se trasladará a la ciudad para capturar.
- Dentro de `doAskBackup()` también tenemos la función `doCapture()`, que primero ejecuta `doQueryPeople()`, función de la práctica anterior, y después manda las peticiones de captura al MTT, mediante un mensaje REQUEST capture name.
- Por último, para finalizar `doAskBackup()`, el agente TS manda un mensaje para cancelar el apoyo al MTT (CANCEL).

En la opción `MOVEBY`:

- Es necesario obtener un agente de tipo DEST que esté en nuestra sesión y que nos envíe su transponder. Para ello, solicitamos al DF todos los agentes DEST y vamos iterando sobre ellos, para pedir el transponder utilizamos la función lstinlinedoAskTransponder().
- Extraemos la ciudad correspondiente del transponder
- Utilizamos los mensajes que ya explicamos en la práctica anterior para utilizar el AUTONAV hasta la ciudad correspondiente.

En la opción `TRANSFER`:

- Utilizamos la función `doTransfer()` para iterar a través de la gente capturada y mandarle una solicitud de captura al DEST al que le pedimos el transponder en el objetivo anterior.

4 Diagrama de clases

Hemos decidido no incluir en la memoria el diagrama de clases, pues al tener un solo agente tenemos una estructura de clases clara, además resulta bastante ilegible. Sin embargo, en el diagrama hay cierta información sobre las funciones del agente que sí que nos gustaría explicar. En general las funciones son las que teníamos en la práctica anterior (excepto la nueva versión de `LARVAblockingReceive()`). Sin embargo, hay otras funciones que hemos añadido:

- `AskForRecharge()`: tal y como ya hemos mencionado, implementa toda la secuencia de mensajes con el DF y con el ajente BB1F necesaria para llevar a cabo la recarga.
- `doQueryPeople()`: hemos incluido una modificación para poder determinar si es necesario añadir la información obtenida en el report general.
- `doAskTransponder()`: es la función que se encarga de buscar un agente de tipo DEST que responda favorablemente a la petición de transponder.
- `doCapture()`: se encarga de iterar a través de las personas que haya en la ciudad y de mandar un mensaje al Session Manager para capturar a cada una de ellas.
- `doTransfer()`: se encarga de iterar a través de las capturas y de mandar un mensaje al agente DEST para transferir a cada una de ellas.