INTELIGENCIA ARTIFICIAL

Proyecto de Cursado – Etapa 2

DESARROLLO DE IA PARA AGENTES EN UN ENTORNO VIRTUAL

2do Cuatrimestre – Año 2017

Integrantes:

Rodrigo Martín Díaz Figueroa – 97400
rodrigomdiaz@live.com

Federico Germán Agra – 94186
federico.agra@gmail.com

ÍNDICE

INTRODUCCIÓN

Descripción de la Etapa

La segunda etapa del proyecto consiste en el diseño e implementación en lenguaje Prolog de un agente inteligente basado en una arquitectura BDI *(Beliefs, Desires, Intentions).* En esta etapa se dotó al agente implementado en la etapa anterior de comportamiento racional considerable, respetando el modelo de agente y la arquitectura BDI. Para ello se llevaron a cabo las siguientes tareas:

- Se definieron los deseos considerados por el agente.

- Se implementó la obtención de todos los deseos actuales y deseos de alta prioridad en base a las creencias del agente.

- Se implementó la selección de una intención a partir del conjunto de todos los deseos actuales.

- Se implementó, para cada intención, la verificación de su cumplimiento.

- Se especificaron acciones del agente (siguiendo la representación Strips) para posibilitar la proyección de planes y detectar así su factibilidad.

- Se estableció e implementó el método de planeamiento para cada tipo de acción de alto nivel.

CORRECCIONES DE ETAPAS ANTERIORES

Actualización de Creencias

Se re-estructuró el modo de actualización de creencias, ya que la implementada en la etapa anterior presentaba inconsistencias en el modo de mantenimiento de los recuerdos y creencias del agente, generando información residuo que imposibilitaba el correcto funcionamiento del agente. Se explicará más adelante los cambios realizados.

*Método de Búsqueda A**

En la versión anterior, a la hora de expandir la frontera se agregaban todos los nodos adyacentes del último nodo visitado, sin verificar que los nodos agregados sean conocidos o no, generando que el algoritmo falle en caso de visitarse algún nodo del que no se tiene ninguna información. Esto, a su vez, producía que en un determinado momento del juego (por lo general, cuando no quedan más objetos por recoger) el agente quedase en un ciclo infinito buscando planes para distintos nodos los cuales

todos fallan. Esto también se debía a que se realizaba un mal cálculo de la función heurística, la cual solo calculaba la distancia a un nodo que contiene a un tesoro.

Método de ordenamiento

En la etapa previa se utilizó un algoritmo de ordenamiento QuickSort para ordenar la frontera en función de su costo más heurística. Se cambió este método por un algoritmo de cola con prioridad, el cual resultó más eficiente, más fácil de implementar, más simple de entender y por ende, más fácil su utilización. Esto último se debe a que al elegir un nodo de la frontera se elige el primero, que es el de menor costo-heurística de todos los nodos (conocidos) de la frontera.

FLUJO DE EJECUCIÓN DEL AGENTE

El agente comienza a ejecutarse una vez que se conecta a la aplicación que gestiona el juego. En ejecución el agente mantendrá activa su rutina de ejecución que consta de 4 partes principales bien definidas:

- Actualización de Creencias.
- Deliberación de Intensiones
- Planeamiento y Ejecución de Acciones
- Realización de Acciones.

Actualización de Creencias

Percepción del Entorno

Se obtiene la percepción del mundo en el instante de tiempo actual, la cual representa una porción del entorno y generada por la propia aplicación. Se identifican en la percepción los siguientes elementos:

- time(T): representa el tiempo de juego en un momento dado, donde T es el tiempo actual.

- node(Id,Pos,Connections): Representa una posición en el mapa del juego, donde Id es el identificador del nodo, Pos es un vector [X,Y,Z] en el espacio y Connections es una lista $[[NodoAdy_1,Costo_1],\ldots,[NodoAdy_n,Costo_n]]$.
  - NodoAdy: Identifica un nodo conectado al nodo Id.
  - Costo: Representa el costo de moverse desde el nodo Id hasta NodoAdy.

- at(Entity,Node): Indica que una entidad se encuentra posicionada en un nodo, donde Entity es un identificador de entidad y Node es el identificador del nodo donde se encuentra.

- atPos/2, atPos(Entity,Pos): Indica que una entidad se encuentra sobre el vector posición, donde Entity es un identificador de entidad y Pos es un vector [X,Y,Z] en el espacio.

- has(Entity1, Entity2): Indica que una entidad lleva consigo otra entidad, donde Entity1 es el identificador de la entidad portadora y Entity2 es el identificador de la entidad que está siendo portada.

- entity_descr(Entity, Descr): Representa una descripción con ciertas características de la entidad que se consideran visibles para el agente, donde Entity es el identificador de una entidad y Descr es una lista de la forma $[[Feature_1,Val_2],[Feature_2,Val_2],\ldots,[Feature_n,Val_n]]$.
  - Feature: Nombre de una característica visible de la entidad.
  - Val: Su valor asociado.
  - Entity: es una entidad del juego de la forma [<entidad>, <nombre>].

Las entidades del juego reconocidas son:

- inn (Posada): estructura utilizada para que el agente se sitúe dentro y regenere su vida hasta el máximo conforme pasa el tiempo.

- agent (Agente): identifica a un agente dentro del juego. Recolecta objetos y ataca a otros agentes.

- gold (Oro): objeto que puede ser recolectado por un agente. No contiene ninguna utilidad.

- grave (Tumba): estructura que almacena oro.

- potion (Poción): objeto que puede ser recolectado por un agente. Sirve para abrir tumbas y dormir a otros agentes.

Mantenimiento de Base de Creencias

Una vez obtenida la percepción actual, se procede a modificar la Base de Creencias del agente. Esta tarea es desarrollada en tres partes: Una parte de *Olvido,* de *Recuerdo* y otra de *Chequeo* de los datos provistos por la percepción. La estrategia de actualización de creencias se basa en borrar, en un principio, toda información alojada en las creencias de la que se ha obtenido una nueva versión en la percepción recibida (*olvido*), para luego recordar todo lo percibido (*recuerdo*), y por último se verifica (*Chequeo*) que lo que se recordaba en el rango de la percepción actual sigue valiendo.

Olvido

La parte de olvido de información obsoleta o desactualizada es la más compleja, ya que se descompone en varias condiciones, en los que en cada uno se eliminan datos recordados según un tipo determinado de relación obtenida en la percepción. Estos tipos de relaciones son:

- Elimina todos los tiempos (*time*) recordados.
- Si se percibe una entidad en el piso que se creía en posesión de otra entidad, se olvida de esa posesión y de cualquier posición (vector) anterior de esa entidad. Por cada *at(E,_)*, si existe en la base de creencias un *has(_,E)* se elimina.
- Si se percibe una entidad en el piso que se creía que estaba en otro lugar del piso, se olvida de la ubicación anterior y de la posición (vector) anterior. Por cada relación *at* (de una entidad) percibido que aparece en la base de creencias, se eliminan todos esos *at* y su correspondiente *atPos.*
- Si se percibe entidad en una posición (vector) que estaba en otra posición (vector), se olvida de la ubicación anterior y de la posición (vector) anterior. Por cada relación *atPos* (de una entidad) percibido que aparece en la base de creencias, se eliminan todos esos *atPos* y su correspondiente *at.*
- Si se percibe entidad en una posición (vector) que estaba poseída por otra entidad, se olvida de dicha posesión. Por cada *atPos(E,_)*, si existe en la base de creencias un *has(_,E)* se elimina.
- Si se percibe una entidad que creía que estaba en el piso y en otra posición (vector) pero en la percepción es poseída por otra entidad, olvida que estaba en el piso y su posición (vector) anterior. Por cada *has(_,E)* encontrado tal que existe en la base de creencias un *at(E,_)* y un *atPos(E,_)*, se eliminan todos los esos *at* y *atPos.*
- Si se percibe que una entidad es poseída por otra, pero que se creía que estaba poseída por una tercer entidad, se olvida de esta última posesión. Por cada *has(_,E)* percibido que existe en la base de creencias, elimina esos *has(_,E).*
- Por cada descripción percibida de una entidad que ya la había recordado antes, olvida su descripción anterior. Se eliminan todas las *entity_descr* que aparezcan en la base de creencias con Entidad igual a la descripción percibida.

Recuerdo

En esta parte lo único que hace es recordar todo lo percibido. Por cada relación obtenida de la percepción, agrega dicha relación.

Chequeo

Por último, se verifica que:

- Para todas las entidades que recordaba en ese nodo (un nodo dentro del rango de percepción del agente) corrobora que sigue estando ahí. Sí, no existe tal relación en la nueva percepción, es decir, que lo que daba por sabido ya no vale, entonces elimino ese recuerdo (*at*).

- Para todas las relaciones *has* encontradas en la percepción, se verifica que esa relación sigue valiendo. Si esta relación ya no se cumple, se elimina de la base de creencias.

Deliberación de Intensiones

Esto se realiza mediante el predicado *deliberate* provisto por la cátedra. El agente analiza si continuará con su intención actual, considerando deseos de alta prioridad, la factibilidad del plan para la intención actual, si la intención actual fue lograda, etc. En caso de no continuar con la intención corriente, establece cual será la nueva intención analizando los deseos existentes y seleccionando uno de ellos.

Planeamiento y Ejecución de Acciones

Realizado mediante el predicado *planning_and_execution(-Action)* provista por la cátedra. Obtiene la siguiente acción primitiva *Action* correspondiente al plan actual, removiéndola del plan. La siguiente acción del plan podría ser de alto nivel (no primitiva), en cuyo caso debe planificarse hasta llegar al nivel de acciones primitivas.

Realización de Acciones

Se informa a la aplicación del juego cual es la próxima acción a realizar. Esta implementado por el predicado *do_action(Action)*.

RAZONAMIENTO Y ESTRATEGIAS

Para que el agente inteligente presente un comportamiento basado en la arquitectura BDI se definieron varias funcionalidades propias de la arquitectura.

Especificación de Deseos

Entre los deseos que un agente puede considerar, se diferencian dos tipos de deseos.

Prioridad Normal

Son aquellas acciones que el agente puede realizar en un momento determinado. Si no hay una acción de alta prioridad ejecutándose, se selecciona, según alguna condición, un deseo de este tipo para continuar. Estos deseos están implementados mediante el predicado *desire(-Desire,-Explanation),* donde *Desire* es un deseo (actualmente activo de acuerdo a las creencias) y *Explanation* es una explicación que especifica las razones por las cuales *Desire* se considera actualmente un deseo. Estos deseos son los siguientes:

- Obtener Poción: *get([potion, <Nombre>]),* si recuerda que una poción se encuentra en el piso, tener esa tumba es una meta.
- Obtener Tesoro dentro de una Tumba: *get([gold, <Nombre>]),* si recuerda que un tesoro se encuentra dentro de una tumba, tener ese tesoro es una meta.
- Obtener un Tesoro tirado en el piso: *get([gold, <Nombre>]),* si recuerda que un tesoro se encuentra en el piso, tener ese tesoro es una meta.
- Obtener un Tesoro depositado en un *Home* ajeno: *get([gold, <Nombre>]),* si recuerda que un tesoro se encuentra en el *Home* enemigo, tener ese tesoro es una meta.
- Descansar: *rest*, se activa este deseo cuando su vida es inferior a 100 pts. Su meta es llegar a una posada para recuperar puntos de vida.
- Depositar Tesoros: *depositarTesoro([gold,<Nombre>]),* su meta es llegar a su *Home* para depositar un tesoro obtenido.
- Defender el *Home: ir_a_casa,* su meta es llegar a su *Home* para defenderlo de posibles ataques de enemigos.
- Mover aleatoriamente: *move_at_random,* su meta es moverse aleatoriamente para mantenerse en movimiento siempre.
- Defenderse: *defenderme_de([agent,UnAgente])*, si hay un agente enemigo en el rango de ataque, decide si quiere huir o atacarlo.
- Explorar: *explorar,* si existen nodos no explorados, su meta es explorarlo.

Prioridad Alta

Son aquellas acciones de suma urgencia que se desean realizar, a tal punto de interrumpir la acción que se está ejecutando si es que ésta no es de prioridad alta. Estos son modelados por el predicado *high_priority(-HPDesire, -Explanation),* el cual determina si existe un deseo *HPDesire* de alta prioridad, es decir, tal que implica abandonar la intención actual. En caso de existir tal deseo de alta prioridad, lo retorna junto a una explicación que especifica las razones por las cuales *HPDesire* se considera un deseo que debe adoptarse inmediatamente como intención. Estos deseos son los siguientes:

- Defenderse: *defenderme_de([agent,UnAgente])*, este deseo se convierte en una intensión de alta prioridad cuando se encuentra un enemigo a menos de 10 unidades de distancia. Su meta es huir si se encuentra en inferioridad de condiciones que su contrincante, o bien atacarlo en caso contrario.
- Descansar: *rest,* Este deseo se convierte en prioridad cuando la vida del agente no supera los 60 pts. Su meta es llegar a la posada más cercana para recuperar puntos de vida.

Selección de Deseos

En esta etapa se determina, según el contexto en el que el agente se encuentra, que deseo elegir como próxima acción a realizar. Esto se modela mediante el predicado *select_intention(-Intention, -Explanation, +CurrentDesires),* el cual selecciona uno de los deseos corrientes del agente como intención. El orden de aparición define el orden de prioridad entre los deseos del agente, seleccionando como intensión al primero en este orden de prioridad.

Orden de Deseos

- Defenderse: Si hay un agente enemigo en el rango de ataque, y su energía es mayor a cero, se elegirá como intensión defenderse de ese agente.
- Descansar: Si el agente tiene el deseo de descansar y tiene menos de 80 puntos de vida, entonces se elegirá como nueva intensión ir a descansar a una posada.
- Obtener poción: Si el agente tiene el deseo de obtener una poción, de todas las posibles pociones tiradas en el suelo, selecciona como intensión obtener aquella que se encuentra más cerca.
- Obtener tesoro de una tumba: Si el agente tiene el deseo de obtener un tesoro escondido dentro de una tumba, de todas las posibles tumbas con objetos, selecciona como intensión obtener aquella que se encuentra más cerca.
- Obtener tesoro del piso: Si el agente tiene el deseo de obtener un tesoro, de todos los posibles tesoros tirados en el suelo, selecciona como intensión obtener aquel que se encuentra más cerca.
- Depositar Tesoros: Si el agente contiene tesoros en su poder, elige como próxima intensión ir a depositarlos en su *Home*.
- Explorar: Si existen nodos que no han sido explorados, entonces elige como intensión ir a explorarlos.

- Descansar: Una vez que ya recogió todos los objetos del mapa, exploró todos los nodos, si tiene le deseo de ir a descansar lo elige como próxima intensión.
- Defender el *Home*: Cuando ya no tienen más que hacer y tiene vitalidad, se elige como próxima intensión ir a situarse cerca del *Home* propio para defenderlos de posibles saqueos enemigos.
- Moverse aleatoriamente: Si agotó todas las posibles acciones anteriores, su próxima intensión será elegir un nodo de manera aleatoria y moverse hacia él.

Verificación de cumplimiento de intensiones

Se verifica mediante el método *achieved(+Intention),* el cual determina si la intención fue alcanzada. Esto es, si se verifica de acuerdo a las creencias del agente.

Listado de verificaciones

- Descansar: Se completa cuando los puntos de vida del agente son los máximos.
- Obtener objeto: Se completa cuando el agente lleva en su poder a ese objeto.
- Ir a posición: Se completa cuando el agente se encuentra en la posición requerida.
- Dejar Objeto: Se completa cuando el agente deja de poseer al objeto en cuestión.
- Depositar Objeto: Se completa cuando el agente y su *Home* se encuentran en la misma posición y el *Home* posee al objeto depositado por el agente.
- Abrir Entidad: Se completa cuando el agente y una entidad se encuentran en la misma posición, y el agente como última acción realizó un *call_spell* sobre dicha entidad.
- Huir: Se completa cuando el agente queda fuera del rango de ataque de un enemigo.
- Atacar: se completa si la última acción del agente es un ataque a un agente.
- Dormir: Se completa cuando la vida del agente enemigo es igual a cero, y la última acción efectuada por el agente (*me*) es un *call_spell* sobre el enemigo.
- Defenderse: Se completa cuando se efectivizan las acciones de atacar a un agente enemigo, o de huir de un agente enemigo o de dormir a un enemigo.
- Explorar: Se completa cuando no quedan nodos desconocidos.

Acciones

Las acciones que puede realizar el agente son las propias acciones primitivas provistas por el juego, además de aquellas acciones de alto nivel generadas por las intenciones. Cada acción de alto nivel se descompone en una secuencia de acciones (que a su vez pueden ser de alto nivel o primitivas).

Primitivas

- pickup([<Tipo>,<Nombre>]): Recoger el objeto de nombre <Nombre> y de tipo <Tipo> (donde tipo puede ser *gold* o *potion*).

- cast_spell(open([grave,<Nombre>],[potion,<NombreP>])): abrir la tumba con nombre <Nombre> lanzando un hechizo con la poción de nombre <NombreP>.
- move([<Posición>]): desplaza al agente hasta la posición <Posición>.
- noop(): no realiza ninguna acción.
- drop([<Tipo>,<Nombre>]): deposita en el piso o en otra entidad el objeto de nombre <Nombre> y de tipo <Tipo>.
- attack([agent,<Target>]): Ataca al agente enemigo de nombre <Target>.

Alto nivel

- explorar:
  - Buscar nodos no explorados.
  - Elegir un nodo no explorado al azar.
  - Ir al nodo elegido.
- get(Obj):
  - Busco la posición del objeto Obj.
  - Ir hasta la posición del objeto Obj.
  - Recoger el objeto Obj.
- get(Obj):
  - Si hay una entidad E que contiene al objeto Obj.
  - abrirEntidad(E).
  - get(Obj).
- ir_a_casa:
  - Buscar la posición del *Home* del agente.
  - Ir hasta la posición del *Home* agente.
- goto(Pos):
  - Buscar un plan de desplazamiento hasta la posición Pos.
- rest:
  - Buscar todas las posiciones de las posadas conocidas.
  - Buscar un plan de desplazamiento para alguna posada Posada encontrada.
  - goto(Posada).
  - stay.
- stay:
  - noop (no hace nada).