Informe IA
Gandolfo, Matias
Marten, Juan

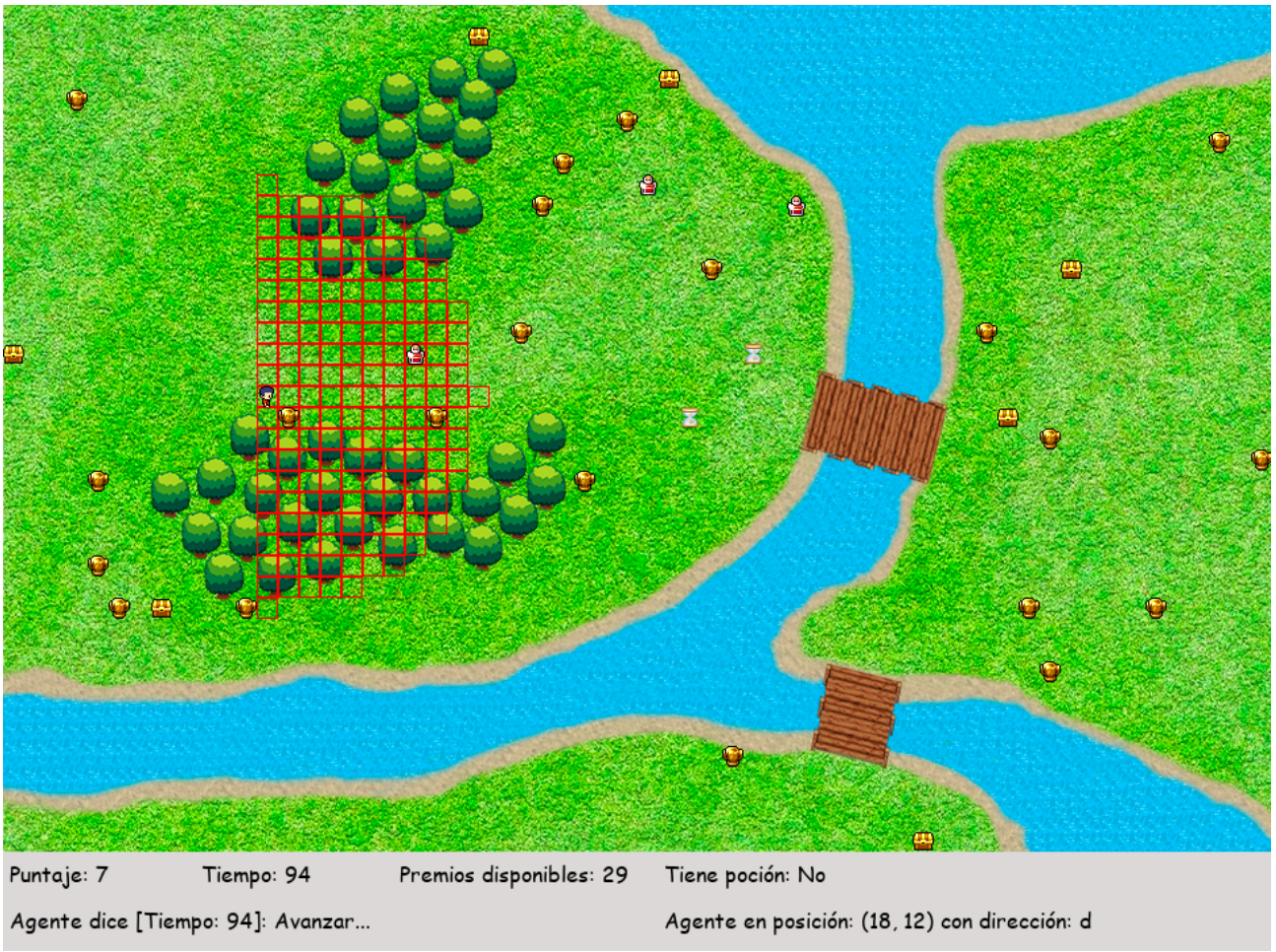
Implementacion de Agente Inteligente

en Prolog

Introducción

El proyecto consiste en la implementación utilizando Prolog, de las funcionalidades de un agente que interactúa con su juego percibiendo su entorno y actuando sobre él mediante una serie de acciones predefinidas. Entre las funcionalidades que se desarrollaron se encuentra la actualización de creencias, en base a la percepción que recibe el agente de su entorno, también se implementó un algoritmo de búsqueda, más específicamente el A* visto en clase, para poder establecer caminos óptimos entre las metas que el agente percibe y en base a estos generar planes de acción. También se realizaron modificaciones en la toma de decisiones del agente, para que no solo utilice el plan de búsqueda creado utilizando A*, sino que también tenga un comportamiento más inteligente.

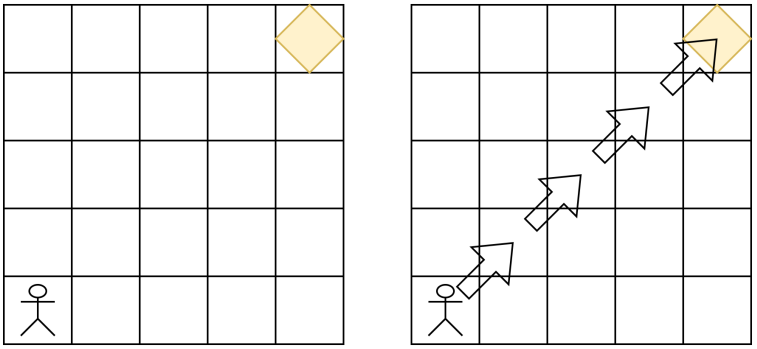
El juego del que forma parte el agente consta de un mapeo con cuadrillas que poseen diferentes costos, no es lo mismo avanzar por el pasto que por el agua, ademas de tambien poseer diferentes objetos, los cuales el agente puede recoger para sumar puntos, aumentar su percepción o aumentar el tiempo de juego restante. El agente no prioriza los objetos según su tipo, únicamente son priorizados por el algoritmo de búsqueda en base a su costo, para encontrar el camino más óptimo.



Aquí se puede ver una imágen ilustrativa del mapa, con sus elementos y el agente cuya percepción son los casilleros en rojo que en este caso está configurado para que pueda detectar hasta diez casilleros por delante de su posición.

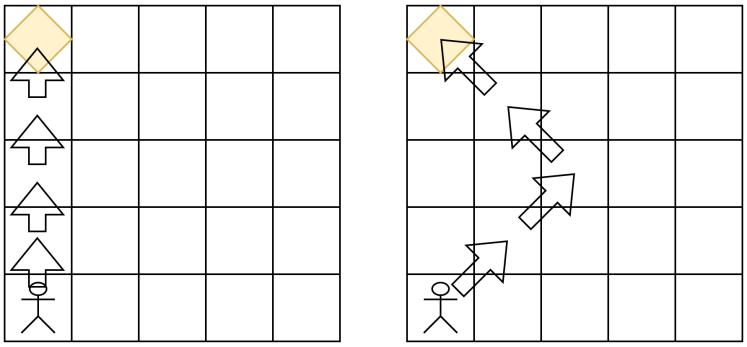
Consideraciones

- Debido a que el costo de mover el agente en diagonal por casillas de costo 1 es menor a la distancia euclídea, usar la distancia euclídea como heurística produce sobreestimaciones. Por lo tanto se reemplazó la función heurística **raíz(dx^2 + dy^2)** por **max(dx, dy)** siendo **dx** la diferencia entre la posición del agente y de la meta en la coordenada **x** y **dy** la diferencia entre la posición del agente y de la meta en la coordenada **y**.



Por ejemplo en este caso la distancia euclídea entre el agente y su objetivo es aproximadamente 6 pero el agente puede llegar a su objetivo en 4 pasos.

- Debido a las consideraciones mencionadas en el ítem anterior, en varios casos es equivalente para el agente desplazarse en línea recta o haciendo movimientos diagonales hacia un lado y luego hacia el otro.



Por ejemplo en este caso el agente puede llegar a su objetivo con el mismo costo por los dos caminos. Debido a esto en algunos casos parece a simple vista que el agente no eligió el mejor camino o la meta más cercana, pero en realidad eligió un camino equivalente o una meta que estaba a la misma cantidad de pasos.

- Para maximizar la cantidad de nodos que el agente percibe mientras se mueve, luego de definir un camino hacia la próxima meta el agente gira para mirar en dirección a la meta.

- Para aumentar la cantidad de nodos que el agente percibe cuando no tiene conocimiento de ninguna entidad en lugar de quedarse quieto haciendo giros de 90° realizará 3 movimientos al azar luego un giro de 180° y repetirá esas acciones hasta percibir una nueva entidad

- Otra funcionalidad adicional del agente, es que aborta el plan cuando en su percepción recibe el nodo vacío en donde anteriormente estaba el objeto meta del plan, luego de hacer esto, recalcula otro plan en caso de que existan objetos restantes en su base de creencias.

Estado interno

Representación de creencias

time/1 → time(T), donde

- T es el tiempo restante de la partida.

node/5 → node(Id, PosX, PosY, Costo, Conexiones), donde

- Id representa un identificador único del nodo, cada nodo es una posición de la grilla con la que se modela el mapa.
- PosX/PosY son las coordenadas en plano cartesiano, correspondientes a (x,y).
- Costo representa el costo individual del nodo, en otras palabras, el costo de avanzar de un nodo adyacente a este.
- Conexiones es una lista de pares [IdVecino, CostoVecino], correspondientes a los nodos adyacentes, donde CostoVecino representa el costo individual de ese nodo.

at/3 → at(IdNodo, TipoEntidad, IdEntidad), donde

- IdNodo es el identificador de un nodo.
- TipoEntidad es un elemento del siguiente conjunto {copa, cofre, poción, reloj, agente}, que nos permite identificar la entidad que se encuentra en el nodo IdNodo.
- IdEntidad es un identificador único de dicha entidad.

direction/1 → direction(D), donde

- D es la dirección actual del agente, y corresponde a una letra del siguiente conjunto {w,s,a,d}.

Actualización y revisión de creencias

La dinámica de actualización de creencias se implementa en module_beliefs_update.pl mediante el predicado update_beliefs(Perc), el cual recibe la percepción del agente representada como una lista de los nodos y entidades que el agente percibe junto al tiempo restante de juego y la dirección actual del agente.

Antes de agregar las nuevas creencias de la percepción a la base de creencias se eliminan las creencias de la posición del agente, su dirección y el tiempo restante. Luego se controlan todas las entidades de la base de creencias actual, en caso de que se perciba la desaparición de una entidad que se encontraba en la base de creencias (el nodo en el que se encontraría la entidad está dentro de la nueva percepción pero la entidad no), dicha entidad se elimina de la base de creencias.

Luego se agregan todos los elementos de la nueva percepción a la base de creencias (si no estaban ya en la base de creencias).

Esta manera de actualizar las creencias cae dentro de la propuesta vista en la materia de mantener la base de creencias libre de contradicciones y es similar a la revisión priorizada del modelo AGM en el caso de que la base de creencias esté compuesta exclusivamente de literales, ya que la base de creencias está compuesta de hechos y cuando llega una nueva creencia que se contradice con las viejas creencias (una entidad que el agente creía que estaba en una posición deja de estarlo), se elimina la creencia anterior para agregar la nueva, manteniendo la consistencia de la base de creencias.

Información adicional guardada en el estado interno

Además de las creencias mencionadas previamente, el agente guarda la siguiente información en su estado interno para mejorar su funcionamiento.

avanzo_random/1

Es utilizado cuando el agente no tiene ningún objeto en su base de creencias, de manera tal que avanza de manera aleatoria 3 veces, luego hace un giro de 180° y reinicia el contador de avanzo_random/1 para repetir estas acciones hasta encontrar un nuevo objeto.

En otras palabras, en el predicado decide_action/2 que hace que el agente gire, se agrega el hecho avando_random(0) y por cada movimiento aleatorio realizado se sumará uno a ese valor. Una vez se realizaron los tres movimientos aleatorios, gira para ver que hay en su espalda y repite.

dest/1

Es un hecho que almacena el ID del nodo destino obtenido a partir de crear un plan. Es utilizado por el agente para poder revisar si en su percepción se sigue encontrando la entidad en el nodo con dicho ID, por lo tanto, si el nodo en el que se encuentra el objeto meta aparece en su percepción y el objeto no, lo eliminará de la base de creencias y al momento de realizar el siguiente paso se dará cuenta que ya no existe, tomando la decisión de generar un nuevo plan a otro objeto.

Comportamiento del agente para tomar decisiones

Una vez que el agente percibe su entorno, toma una decisión utilizando el predicado decide_action/2. Este predicado contempla como posibles acciones:

- Levantar un objeto, esta situación se da cuando el agente se encuentra en la misma posición que un objeto. El objeto puede ser un elemento del conjunto {copa, cofre, poción, reloj}.
- Seguir las instrucciones del plan, si el agente tiene un plan guardado actualmente consiste en avanzar hacia el siguiente nodo del camino indicado en el plan.
- Crear un plan, si el agente no tiene un plan guardado y tiene al menos un objeto alcanzable en su base de creencias, busca entre sus metas utilizando el algoritmo de búsqueda A* y a partir del camino devuelto por el algoritmo se genera un plan.

    El plan se crea utilizando el predicado busqueda_plan/3 que calcula los nodos meta de la base de creencias y llama a buscar_plan_desplazamiento/4, el cuál busca un camino desde el nodo actual del agente hasta el mejor nodo meta utilizando el algoritmo A* (buscarEstrella/5) y crea un plan de desplazamiento en base al camino retornado por buscarEstrella/5.
    Cuando se crea un plan, se utiliza la meta obtenida para girar el agente en dirección al destino y maximizar los nodos percibidos mientras camina hacia dicho destino.

- Movimiento aleatorio, en caso de que no tenga ningún objeto alcanzable en su base de creencias para reconocer terreno y así poder encontrar nuevos objetos.
- Girar 180° luego de haber realizado 3 movimientos aleatorios para poder obtener una percepción de los nodos a la espalda del agente.

Estrategia de búsqueda A*

El predicado buscarEstrella/5 es el que inicia el proceso de buscar un camino para llegar a una de las posibles metas, toma como entrada una frontera con el nodo inicial y un conjunto de metas. Como resultado, retornará la meta cuyo costo de llegar es el menor del conjunto de metas, el camino a esa meta y el costo del camino.

El primer predicado invocado por buscarEstrella/5 es buscar/4 que recibe una frontera, ordenada en base a la función h(X), una lista de visitados que permitirá evitar ciclos y no volver a visitar nodos ya visitados, la lista de metas que se utilizarán para detectar si llegamos a un nodo meta, además de ser necesarias para calcular la función h(X) y retornará el destino, es decir, la meta más óptima para el agente. buscar/4 se encargará de recorrer la frontera tomando el primer nodo, que corresponde al que menor h(X) posee, controlar si es una meta, en caso de serlo lo retorna y en caso de no serlo obtiene los nodos adyacentes, los agrega a la frontera (en caso de que no haya sido visitado ni haya sido agregado a la frontera por un mejor camino) y vuelve a llamar a buscar/4 recursivamente. Al agregar los nodos vecinos a la frontera además se realiza un assert de padre(IdPadre, IdVecino) para poder reconstruir el camino hacia la meta.

La siguiente llamada es a encontrarCamino/2 que recibe el destino que devolvió buscar/4 y retorna el conjunto de nodos por los que el agente debe avanzar para llegar a la meta, utilizando los hechos padre-hijo para reconstruir el camino.

Luego utilizando el predicado costoCamino/2 obtendrá el costo de recorrer el conjunto de nodos que corresponden al camino que lleva al agente desde su posición original a la meta.

Llegado a este punto buscarEstrella/5 ya tiene todos los valores necesarios para retornar el camino con su costo y destino.

Preguntas

¿Qué otro algoritmo podría utilizarse para encontrar un camino hacia los tesoros?

Los métodos ciegos podrían ser prohibitivos en cuanto a uso de recursos, además de que no todos garantizan encontrar una solución óptima.

Con respecto a los métodos de búsqueda heurísticos, Best-FS encontraría la meta más cercana pero el costo del camino que recorrería podría ser muy malo, ya que no realiza un chequeo de costos al seleccionar el próximo nodo a visitar.

HDFS también lograría encontrar la meta más cercana, sin tener en cuenta el costo del camino que recorre.

¿Qué ocurriría respecto a los objetos en el caso de utilizar un método de búsqueda ciega, en lugar de A*?

DFS recorrería el espacio de búsqueda en profundidad hasta encontrar una meta cualquiera, la cuál dependerá del orden en el que se hayan definido los nodos, por lo que no se asegurá que encuentra la meta más cercana ni que el camino sea óptimo.

BFS recorrería el espacio de búsqueda por niveles, por lo que siempre encontrará la meta más cercana respecto a cantidad de pasos pero el costo del camino podría no ser óptimo.

LCFS recorrería los nodos del espacio de búsqueda en función de cuál tiene menor costo, por lo que siempre encontraría la meta a la que puede llegar con menor costo, es decir que el camino sería óptimo.

¿Cómo se comportaría el agente respecto a los obstáculos en caso de buscar un plan de desplazamiento utilizando un método de búsqueda ciega?

DFS y BFS no consideran el costo de los nodos cuando eligen el próximo nodo a visitar de la frontera, por lo que atravesarían los obstáculos ignorando que son obstáculos.

LCFS elige los nodos con menor costo total, por lo que primero exploraría los caminos que eviten los obstáculos (mientras que el costo total del camino sea menor al costo de atravesar el obstáculo). Por lo tanto evitará los obstáculos a menos que el mejor camino posible sea atravesando dicho obstáculo.

¿Sería posible, o adecuado, utilizar un método de búsqueda que no almacene la frontera, como por ejemplo Hill Climbing?

No sería adecuado utilizar estos métodos para encontrar un camino a la meta ya que estos métodos se centran en encontrar una meta en lugar de un camino a la meta, y en este problema las metas ya son conocidas. Además podrían estancarse sin encontrar la meta.

En este caso particular, por cómo fue definida la heurística, no podría estancarse ya que no hay mínimos locales por lo que podría adaptarse el algoritmo de hill climbing para que guarde el camino recorrido hacia la meta más cercana y luego pueda devolverlo con un bajo uso de memoria, ya que, no debe almacenar la frontera. Aunque el camino podría no ser óptimo.