Práctica 2 TSI

Miguel Lentisco Ballesteros

Ejercicio 1

Como es el ejercicio base de todos, explico los distintos apartados básicos que he realizado.

Types

Primero definimos los tipos esenciales y sus subtipos que vamos a representar, teniendo en cuenta que podrá haber diferentes objetos de cada subtipo de objeto o varios jugadores, etc.

- Zona (representa una zona del mapa)
- Objeto (representa los distintos objetos que hay)
  - Oscar
  - Manzana
  - Algoritmo
  - Oro
  - Rosa
- Npc (representa un personaje no jugable)
  - Leonardo
  - Profesor
  - Princesa
  - Principe
  - Bruja
- Persona (representa a personajes y jugadores)
  - Npc
  - Player (representa un jugador)
- Posicionable (representa cualquier cosa que pueda "estar" en una zona)
  - Persona
  - Objeto

Predicates/functions

Representamos los distintos estados y relaciones que tiene que haber:

Predicates:

- Que una cosa está en cierta zona: `(estaEn ?p - Posicionable ?z - Zona)`
- Que una persona tiene cierto objeto: `(tiene ?p - Persona ?o - Objeto)`
- Asumimos que un jugador solo puede tener cogido un objeto a la vez y que los personajes puedan recibir objetos sin límites.
- Que un jugador tenga la mano llena: `(manoLlena ?j - Player)`

Functions:

- Representamos la orientación como un valor númerico siendo 0 el Norte, 1 Este, 2 Sur, 3 Oeste; hago esto para evitar tener varios predicados de orientación y tener un tipo de orientación.
- La orientación de zonas, considerando donde está z2 respecto de z1: `(conectadas ?z1 - Zona ?z2 - Zona)`. Considero que no es una relación simétrica pues pudiera ser que de una zona se pueda ir a otra pero no al revés además de que complica más las condiciones de las acciones.
- La orientación de un jugador: `(orientado ?j - Player)`

Actions

Tenemos que entregar objetos en zonas a distintos personajes, para ello necesitamos movernos (`moverseA, girarDerecha, girarIzquierda`), interactuar con objetos (`cogerObjeto, dejarObjeto`) y finalmente entregar el objeto al personaje (`entregarObjeto`).

- `girarDerecha/Izquierda` : cambia el valor númerico de la orientación del jugador según toque.
- `moverseA` : el jugador debe estar en z1 y debe coincidir la orientación de z1 y z2; el jugador acaba en z2.
- `cogerObjeto` : el jugador no debe tener la mano llega y estar con el objeto en la misma zona, coge el objeto y tiene la mano llena.
- `dejarObjeto` : el jugador deja en la zona donde está un objeto que tenga en la mano, y ya no tiene la mano llena.
- `entregarObjeto` : el jugador entrega al personaje en la misma zona un objeto que tenga, teniendolo ahora el personaje y el jugador teniendo la mano vacía.

Goal

El objeto que ofrece el parser es entregar objetos a los personajes de manera que al leer los objetos y personajes relaciona para cada personaje un objeto aleatorio (por ello en el mapa debe haber al menos tantos objetos como personajes).

Problemas

He incluido dos problemas, uno de ejemplo pequeño (7 zonas - 2 objetos - 2 personajes) y otro mas grande (25 zonas - 7 objetos - 7 personajes), además el problema2 incluye varios objetos y personajes con el mismo tipo para probar que sigue funcionando.

Parser

En general para todos los ejercicios asumo que todos los objetos, personajes y jugadores se incluyen en los corchetes de la primera vez que aparezca una zona así como el tipo de la zona, si sale otra vez la zona con otra cosa en los corchetes se ignora.

Ejercicio 2

Modificaciones

En este ejercicio tenemos que considerar que hay un coste de ir de una zona a otra, y además incluiremos que se pueda minimizar el plan según la distancia recorrida, para ello dejamos todo igual que en el Ejercicio 1 añadiendo las siguientes functions:

- `(distanciaJugador ?j - Player)` : la distancia recorrida por un jugador (inicializada a 0).
- `(distanciaTotal)` : la distancia total recorrida por todos los jugadores.
- `(distanciaZona ?z1 - Zona ?z2 - Zona)` : la distancia de una zona a otra (coste del camino).

Finalmente modificamos la acción `moverseA` para que incremente la distancia del jugador con el coste del camino e igual con la distancia total.

Además añadimos en los ficheros de problemas para minimizar distanciaTotal.

Problemas

Son los mismos que el Ejercicio1, añadiendo distintos valores de peso entre los caminos, y así ejecutamos sin optimización (Plan_problema1.txt, Plan_problema3.txt) y con optimización (Plan_problema2.txt, Plan_problema4.txt) los problemas 1 y 2 respectivamente. Obtenemos en los optimizados planes más largos que los sin optimizar pero obtenemos una minimización de la distanciaTotal recorrida.

Ejercicio 3

Modificaciones

En este ejercicio ahora cada zona tiene un tipo distinto de terreno, de manera que no se puede ir a una zona `Precipicio`, y para ir a una zona `Bosque` o `Agua` necesitamos de una `Zapatilla` o `Bikini` respectivamente.

Para ello añadimos los subtipos `Bosque Agua Precipicio Arena Piedra` al supertipo `Zona`, además hacemos los subtipos `ObjetoUsable` y `ObjetoEntregable` del supertipo `Objeto` de manera que `Bikini Zapatilla` sean del tipo `ObjetoUsable` y `Oscar Rosa Algoritmo Oro Manzana` de `ObjetoEntregable`; haciendo esto para que podamos definir en las acciones el entregar solo objetos entregables, y usar solo objetos entregables.

Además necesitamos saber si un objeto es zapatilla o bikini, al igual que los tipos de zonas; se podia haber definido constantes para los tipos de zona y no haber puesto subtipos y así usar un predicado del tipo `(esZona ?z - Zona ?t - Tipo)` pero para futuras modificaciones que requieran tratar a alguno de los subtipos en concreto puede que sea beneficiable tener está subdivisión; en cualquier caso cambiar eso sería modificar el parser muy levemente, así que he preferido tenerlo así para hacerlo por subdivisión de tipos `(esBosque ?z - Zona)` etc. En el caso de los objetos usables hay que usar un `esZapatila(?o - Objeto)` y `esBikini(?o - Objeto)` si o sí, ya que al intentar usar algun tipo de constante puede que colisione con un nombre de objeto.

Ahora contamos con mochila (1 hueco) por lo que se añadimos predicates `(enMochila ?j - Player ?o - Objeto)` y `(mochilaLlena - ?j Player)` y se añaden las acciones `sacarDeMochila` y `meterEnMochila`; para sacar el objeto de la mochila necesitamos tener la mano vacía y para meter un objeto en la mochila que tenga el jugador en la mano debe tener la mochila vacía.

Finalmente modificamos el método `moverseA` (ahora tiene un objeto usable como parametro) para que si es un precipicio no pueda moverse, y si es bosque o agua necesita tener una zapatilla o un bikini respectivamente o en la mochila o en la mano.

Como apunte ahora el método `entregarObjeto` solo funciona con objetos de tipo `ObjetoEntregable`.

Problemas

Parecidos a los problemas del Ejercicio2, añadiendo los distintos terrenos y objetos de tipo Zapatilla y Bikini de manera que al menos una zapatilla está en Agua o un bikini está en Bosque.

En el problema 1 por ejemplo tiene que forzosamente ir a una zona de agua y para ello necesita el bikini que está en zona bosque así que acaba llendo a por una zapatilla en piedra.

Igual pasa en el problema 2, solo que a una dimensión más grande; forzando así a que encuentre un plan que tenga que tener en cuenta ir cogiendo y dejando el bikini/zapatilla para poder cambiar de terrenos e ir cogiendo los objetos y entregando a los personajes.

Ejercicio 4

Modificaciones

En este ejercicio ahora cambiamos el objeto, ya no vamos a entregar X objeto a Y personaje, ahora cada vez que un jugador entrega un objeto gana cierta cantidad de puntos y queremos que se consigan en total por todos los jugadores una cierta cantidad.

Primero definimos functions del tipo `(puntosJugador ?j - Player)` y `(puntosTotales)` para llevar la cuenta de los puntos totales, y además definimos `(daPuntos ?p - Npc ?o - ObjetoEntregable)` de manera que así representamos la cantidad de puntos que da un npc por recibir cierto objeto entregable.

Así modificamos la acción `entregarObjeto` para añadir los puntos `(daPuntos ?p ?o)` a `(puntosJugador ?j)` y `(puntosTotales)`

Parser

Se cambia ahora la goal a que los puntos totales superen cierta cantidad, y se añaden los valores de `(daPuntos ?p ?o)` según la tabla que se daba.

Problemas

Los de Ejercicio3, haciendo que en el primer problema tenga que entregar una combinación específica (manzana a la bruja y óscar a la princesa) y no cualquier otra combinación que hace que no llegue; en el problema 2 tenemos una cantidad máxima de puntos posible (100p - 10 objetos) y ponemos una puntuación a obtener de (85p) para que este forzado a entregar bien casi todos los objetos a los personajes correspondientes.

Ejercicio 5

Modificaciones

En este ejercicio nos dicen ahora que cada personaje ahora tiene un bolsillo y solo puede recibir cierta cantidad de objetos, entonces definimos functions `(nBolsillo ?p - Npc)` que representa el nº de objetos que tiene actualmente y `(maxBolsillo ?p - Npc)` que representa el nº máximo de objetos que pueda tener el bolsillo.

Finalmente modificamos la acción `entregarObjeto` con precondición añadida `(> (maxBolsillo ?p) (nBolsillo ?p))` y al realizar la entrega aumentamos en 1 el nº de objetos que tiene en el bolsillo con `(increase (nBolsillo ?p) 1)`

Problemas

Usando los problemas anteriores, en el primero tenemos 2 manzanas y una rosa, dos brujas y una princesa, objetivo 30p; de manera que las dos manzanas se encuentran en el mismo lugar que la bruja haciendo que lo más fácil sea entregarle las dos manzanas a la bruja, pero ponemos que cada personaje tengan bolsillo de 1 objeto nada más, haciendo que tenga que ir a entregar la manzana a la otra bruja.

En el segundo problema se plantea la misma situación con varios personajes y varios objetos, forzando a que tenga que cambiar de personaje para entregar el objeto.

Ejercicio 6

Modificaciones

En este ejercicio nos dicen ahora que hay 2 jugadores cooperando en el mapa, debido a como habiamos definido todo en general para distintos jugadores no tenemos que modificar nada para que funcione. No añado intercambio de objetos como acción ya que se puede realizar ya dejando objeto y recogiendo (y no se pide en el ejercicio esta acción).

Parser

Ahora el parser añade en el goal aparte del `(puntosTotales)` los `(puntosJugador ?j)` minimos que tiene que tener cada jugador.

Problemas

Usando los problemas anteriores añadimos 2 jugadores en cada mapa y los ponemos para que cada uno consiga una cantidad mínima de puntos, dejando además solo una zapatilla y un bikini para que tengan que cooperar y dejarse entre sí los objetos.

He bajado el nº de puntos a obtener en el segundo problema para que no tarde tanto la búsqueda y no se quede colgado el programa.

Ejercicio 7

Modificaciones

En este ejercicio nos dicen que ahora hay 2 tipos de jugadores Dealer y Picker de manera que el Dealer es el que entrega los objetos a los personajes pero no puede recogerlos del suelo mientras que el Picker es al revés, coge los objetos de suelo pero no puede entregarlos a los personajes. Así tienen que cooperar ambos jugadores de manera que el picker recoge los objetos y se los da al dealer para que pueda entregarlos y ganar puntos.

Añadimos los dos subtipos `Picker` y `Dealer` del supertipo `Player`, y ponemos restricciones en los tipos de los parámetros de las acciones (`entregarObjeto` ahora debe ser Dealer y `cogerObjeto` debe ser Picker) y hacemos otra acción que sea `darADealer` para que le de el objeto del picker al dealer.

Finalmente cambiamos la function `(puntosJugador ?p - Dealer)` para que sea explícitamente un dealer.

Parser

Ahora solo se añade al goal `(puntosJugador ?j)` a los dealer.

Problemas

Usando los problemas anteriores ponemos un dealer y un picker para que tengan que cooperar entre ellos para poder superar el mapa.

Igual usamos un problema medio sencillo en el segundo programa para que resuelva en un tiempo moderado.