Alberto Olcina Calabuig Alma Salmerón Sena

PRÁCTICA 1: PYGOMAS

Planteamiento

El comportamiento esperado de este agente es el siguiente:

- Desde la posición de aparición, debe dirigirse al centro de la arista más cercana
- Una vez llega a esa posición, barrerá todo su campo de visión
- En caso de que detecte a alguien, dejará de girar y disparará hasta matarlo o hasta que su vida (o número de balas) sea menor que un mínimo establecido
- En caso de que se rebase este mínimo, el agente se dirigirá al centro
- Una vez en el centro, buscará paquetes de medicina o balas, y recargará hasta superar otro mínimo
- Cuando haya recargado buscará el centro de la arista más cercana y repetirá el proceso.

Código en ASL

A partir de la creencia flag (F) se lanza un plan cada vez que se ejecuta el programa.

En este se utiliza un test goal para recuperar la posición del agente, ese valor se pasa como parámetro a una función getNearestCentralLimit, que calculará cuál es la arista más cercana a ese punto y se dispara una acción goto que hace al agente ir a ese punto.

```
+flag(F): team(200)
  <-
  ?position(G);
  .getNearestCentralLimit(G, P);
  .goto(P).
```

El plan que define el comportamiento del agente cuando su vida es baja se lanza a partir de la creencia health (H) y se activa cada vez que la vida del agente baja de 30.

Se suprime la creencia giro (la que hace que el agente barra todo su campo de visión cuando está en la arista), se añade la creencia center (que indica que el agente debe ir al centro) y se dispara una acción goto que hace que el agente vaya al centro.

El plan basado en la creencia ammo (A) funciona igual que el de la creencia health (H), pero comprobando si es el número de balas el que baja de 30.

```
+health(H): H < 30
  <-
  -giro;
  +center;
  .goto([128, 0, 128]).
```

```
+ammo(A): A < 30
  <-
  -giro;
  +center;
  .goto([128, 0, 128]).
```

Para comprobar que el agente ya ha recargado, se lanzan otros dos planes. En ellos, se comprueba que, estando en el centro (creencia center) tanto la vida como el número de balas son mayores que 75.

En caso de que ambos parámetros superen el mínimo, se elimina la creencia center, se vuelve a obtener la posición del agente, se computa cuál es el centro de la arista más cercana y se lanza una acción para ir hasta él.

```
+health(H): H > 75 & center & ammo(A) & A > 75
  <-
  -center;
  ?position(G);
  .getNearestCentralLimit(G, P);
  .goto(P).

+ammo(H): H > 75 & center & health(A) & A > 75
  <-
  -center;
  ?position(G);
  .getNearestCentralLimit(G, P);
  .goto(P).
```

Cuando el agente detecta un paquete en su campo de visión lanza un plan para ir a por él. Para ello se comprueba que no esté yendo a por ningún otro paquete usando la creencia not(gettingPack) y comprobará o bien que la vida sea menor que 100 si se encuentra un paquete de salud (tipo 1001), o bien que el número de balas sea menor que 75 si el paquete es de munición (tipo 1002).

En ambos casos, añadirá la creencia gettingPack y lanzará una acción goto para ir a la posición del paquete detectado.

```
+packs_in_fov(ID, Type, Angle, Distance, Health, Position) : not(gettingPack) & ammo(A) & A < 75 & Type == 1002
  <-
  +gettingPack;
  .goto(Position).

+packs_in_fov(ID, Type, Angle, Distance, Health, Position) : not(gettingPack) & health(H) & H < 100 & Type == 1001
  <-
  +gettingPack;
  .goto(Position).
```

Cuando se recoge el paquete se lanza un plan pack_taken, que elimina la creencia gettingPack.

```
+pack_taken(N, T): team(200)
  <-
  -gettingPack.
```

Cuando se llega a un punto, se lanza un plan target_reached. Para evitar que interfiera con los casos en los que se están buscando paquetes de salud o munición, se comprueba que la creencia center no está activa.

Este plan llama a la función getInitialLookAt, que calcula a qué punto debe mirar el agente para que la pared quede a su espalda, y dispara una acción look_at que hace que el agente mire a ese punto. A continuación, añade la creencia giro, lanza el plan turn_AroundA y elimina el plan target_reach (es decir, se elimina a sí mismo porque sólo debe lanzarse una vez).

```
+target_reached(T): not(center) & team(200)
  <-
  .getInitialLookAt(T, L);
  .look_at(L);
  +giro;
  +turnAroundA(0);
  -target_reached(T).
```

Los planes turnAroundA, turnAroundB y turnAroundC son análogos entre ellos, activándose cuando está activa la creencia giro y no está la creencia enemiesDetected.

Estos planes cogen la posición en la que se encuentra el agente, esperan un pequeño margen de tiempo y lanzan uno de los tres métodos que les indican cómo deben girar. El punto obtenido con estas funciones se pasa como parámetro a la acción look_at, se vuelve a esperar un pequeño margen de tiempo, se añade el plan que continuará el giro y se elimina el propio plan.

Si estamos en el turnAroundA se pasará al turnAroundB, si estamos en el turnAroundB se pasará al turnAroundC, y si estamos en el turnAroundC se pasará al turnAroundA para reiniciar el giro.

```
+turnAroundA(B) : giro & not(enemiesDetected)
  <-
  ?position(P);
  .wait(500);
  .lookSideOne(P,M);
  .look_at(M);
  .wait(100);
  +turnAroundB(0);
  -turnAroundA(B).
```

```
+turnAroundB(C) : giro & not(enemiesDetected)
  <-
  ?position(P);
  .wait(500);
  .getInitialLookAt(P, N);
  .look_at(N);
  .wait(100);
  +turnAroundC(0);
  -turnAroundB(C).

  +turnAroundC(D) : giro & not(enemiesDetected)
  <-
  ?position(P);
  .wait(500);
  .lookSideTwo(P, Q);
  .look_at(Q);
  .wait(100);
  +turnAroundA(0);
  -turnAroundC(D).
```

A continuación, hay algunos métodos que no se usan en esta versión del agente (los planes patroll_point y enemies_in_fov).

Por último, el plan friends_in_fov detectará cuando alguien entra en el campo de visión del agente. Cuando eso ocurra, en caso de tener vida y munición suficiente, añadirá la creencia enemiesDetected, disparará 20 balas y lanzará un plan auxiliar noEnemies, que comprobará si ya se ha matado al agente que haya entrado en el campo de visión.

En caso de que pasado un periodo de tiempo no se detecte ningún agente, eliminará la creencia enemiesDetected.

```
+friends_in_fov(ID,Type,Angle,Distance,Health,Position) : ammo(A) & A > 0 & health(H) & H > 30
  <-
  +enemiesDetected;
  .shoot(20,Position);
  +noEnemies(0).

+noEnemies(E) : enemiesDetected
  <-
  .wait(400);
  -enemiesDetected.
```

Código en Python

Se han definido algunas funciones extra que se llaman desde los planes.

En primer lugar, la función getNearestCentralLimit calcula cuál es la arista más cercana a la posición del agente. Para ello se definen cuatro puntos (first, second, third y fourth) con los centros de cada arista. Luego, se calcula la distancia de Manhattan respecto a dichos puntos desde la posición actual del agente y se selecciona aquel punto cuya distancia respecto a la posición actual sea menor (no se considera la coordenada y, puesto que esta es cero para todos los puntos posibles).

Si la distancia es menor que el mínimo ya obtenido anteriormente (minDist), se actualiza su valor y el de goTo. Finalmente, se devuelve la variable goTo, que será igual a uno de los cuatro puntos inicialmente definidos.

```
@actions.add_function(".getNearestCentralLimit", (tuple))
def _getNearestCentralLimit(pos) :
    x, y, z = pos[0], pos[1], pos[2]
    first = [121,0,10]
    second = [121,0,244]
    third = [10,0,121]
    fourth = [244,0,121]
    listP = [first, second, third, fourth]
    minDist = 999999
    goTo = None
    for point in listP :
        distance = abs(point[0] - int(x)) + abs(point[2] - int(z))
        if (distance < minDist) :
            minDist = distance
            goTo = point
    goTo = tuple(goTo)
    return goTo
```

También se definen tres funciones similares entre ellas getInitialLookAt, lookSideOne y lookSideTwo, que definen los puntos a los que debe girar el agente según su posición. Para ello se extraen los valores *x, y* y *z* de la posición que se pasa como parámetro.

En el método getInitalLookAt, si el valor de x es 121 (se encuentra en el centro de la arista de la izquierda o la derecha), se usa el valor de z para saber en qué lado está. Si está en la izquierda, el agente se gira hacia la derecha para dejar la pared a su espalda y viceversa.

Si el valor de z es 121 (se encuentra en el centro de la arista superior o inferior), se usa el valor de x para saber en cuál de ellas está. En caso de que esté arriba se gira hacia abajo y viceversa.

Las funciones lookSideOne y lookSideTwo replican este comportamiento, pero en lugar de mirar en dirección contraria y perpendicular a la pared, lo hacen con un pequeño ángulo.

En caso de que el agente esté en la arista superior o inferior, esto se hace variando ligeramente el valor de la coordenada x en el punto que se devuelve (respecto al de getInitialLookAt). Si está en una de las aristas laterales, se varía el valor de z.

En estas tres funciones los puntos a los que el agente debe mirar están predefinidos y, como depende únicamente de su posición, no se calcula ninguna distancia.

Al igual que en el método getNearestCentralLimit, se devuelve una tupla que se usará como entrada para ejecutar otras acciones.

```
@actions.add_function(".getInitialLookAt", (tuple))
def _getInitialLookAt(pos) :
    x, y, z = pos[0], pos[1], pos[2]
    lookAt = []
    if ((x == 121) and (z == 10)):
        lookAt = [121,0,25]
    elif ((x == 121) and (z == 244)):
        lookAt = [121,0,230]
    elif ((x == 10) and (z == 121)):
        lookAt = [25,0,121]
    elif ((x == 244) and (z == 121)):
        lookAt = [230,0,121]

    lookAt = tuple(lookAt)
    return lookAt
```

```
@actions.add_function(".lookSideOne", (tuple))
def _lookSideOne(pos) :
    x, y, z = pos[0], pos[1], pos[2]
    lookAt = []
    if ((x == 121) and (z == 10)):
        lookAt = [90,0,25]
    elif ((x == 121) and (z == 244)):
        lookAt = [90,0,230]
    elif ((x == 10) and (z == 121)):
        lookAt = [25,0,90]
    elif ((x == 244) and (z == 121)):
        lookAt = [230,0,90]

    lookAt = tuple(lookAt)
    return lookAt
```

```
@actions.add_function(".lookSideTwo", (tuple))
def _lookSideTwo(pos) :
    x, y, z = pos[0], pos[1], pos[2]
    lookAt = []
    if ((x == 121) and (z == 10)):
        lookAt = [150,0,25]
    elif ((x == 121) and (z == 244)):
        lookAt = [150,0,230]
    elif ((x == 10) and (z == 121)):
        lookAt = [25,0,150]
    elif ((x == 244) and (z == 121)):
        lookAt = [230,0,150]

    lookAt = tuple(lookAt)
    return lookAt
```