UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICAS DE SISTEMAS INTELIGENTES

DESARROLLO DE UNA INTELIGENCIA ARTIFICIAL

- WILLY -

Autores:

José Ignacio Díaz Marmolejo

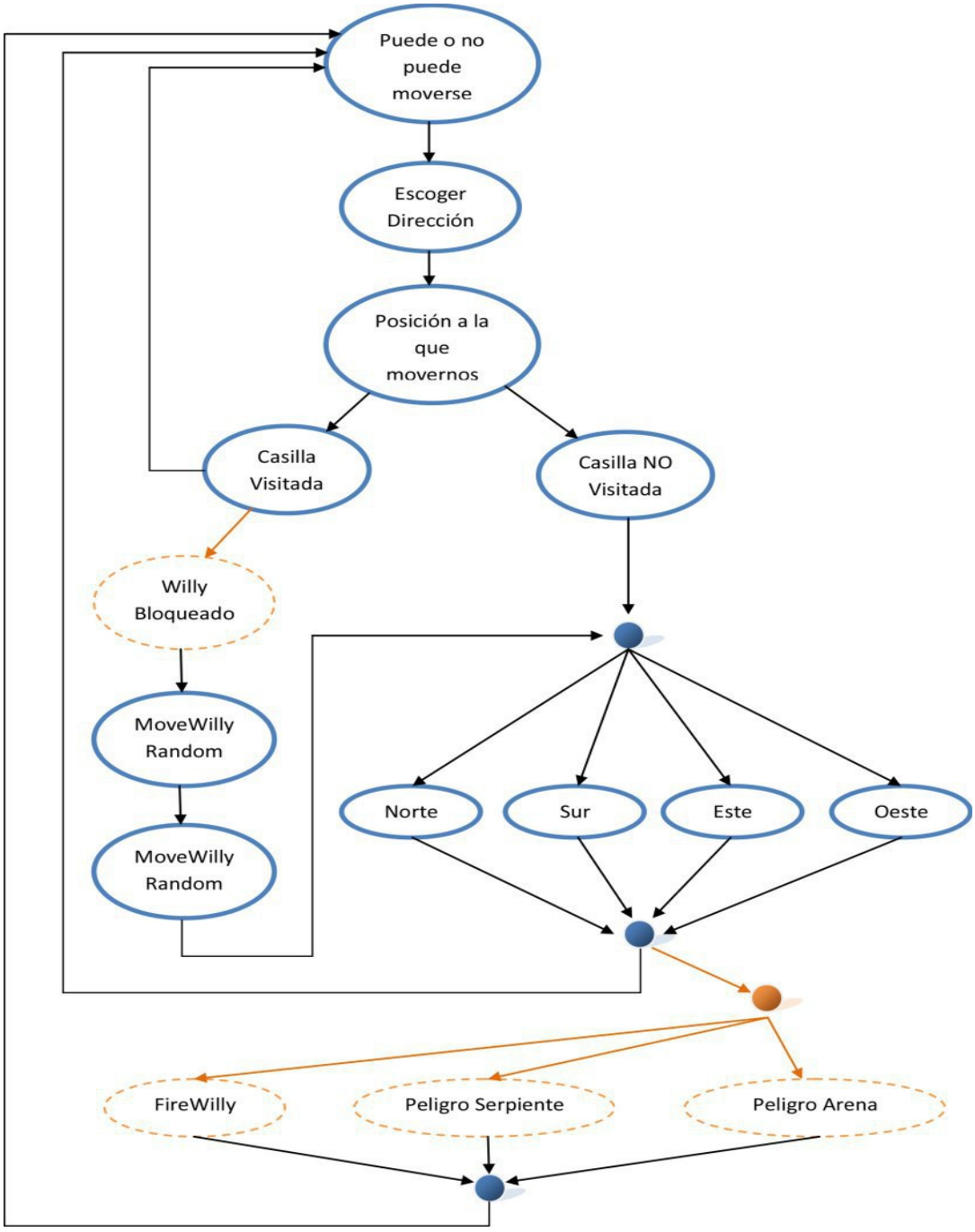
Diego Llamas Nuflo

Fecha:

11-05-2017

CAPÍTULO 1

Diagrama de estados



CAPÍTULO 2

Definición de reglas y hechos

2. 1 Funcionamiento

EL programa se ejecuta bajo una serie de reglas previamente definidas, ilustradas en el capítulo 1, dando información sobre el orden de cómo se ejecutarían. En función de los distintos estados que vaya tomando Willy, se ejecutará una regla u otra, siendo algunas de ellas más importantes.

- Los distintos estados del diagrama mostrado en el capítulo 1, de color **negro** indican el camino optimo (Happy path) que debería seguir Willy para llegar a su destino.
- Los distintos estados del diagrama mostrado en el capítulo 1, de color naranja indican que son reglas que tienen prioridad sobre otras.

2. 2 Definición de reglas y hechos

2. 2. 1 deffacts

```
(deffacts Constructor
        (ultimoMov NULL)
        (movimientos 0)
        (casilla 0 0)
        (casilla-actual 0 0)
        (Acto 0)
        (contador 0)
)
```

El deffacts contiene todos los hechos que se introducirán desde el principio. Éstos serán los que necesiten las reglas para

ejecutarse o los que establecerán la dinámica de ejecución del programa.

Son los siguientes:

- ultimoMov: hecho donde guardaremos la dirección del último movimiento realizado. Hecho único.
- movimientos: hecho donde guardaremos el número de movimientos. Hecho único.
- casilla: hecho que guarda una casilla ya visitada. Se repetirá durante la ejecución con nuevas casillas.
- casilla-actual: hecho que guarda la posición donde se

2. 2. 2 defrule

2. 2. 2. 1 defrule Puede-o-NoPuede-Moverse

```
(defrule Puede-o-NoPuede-Moverse
        (declare (salience 50))
        ?id <- (Acto 0)
        (movimientos ?x)
        (test (< ?x 999))
=>
        (retract ?id)
        (assert (Acto 1))
)
```

Esta función permite establecer un límite de movimiento sobre el que Willy no se podrá mover.

2. 2. 2. 2 defrule EscogerDirección

```
(defrule EscogerDireccion
        ?id <- (Acto 1)
        (movimientos ?x)
```

```
        (directions $? ?direction $?)
    =>
        (retract ?id)
        (assert (Acto 2))
        (assert (proxMov ?direction))
    )
```

Esta función escoge una dirección al azar sobre la cual se hará un breve análisis para ver si es posible moverse a dicha dirección

2. 2. 2. 3 defrule Posición-a-la-que-Movernos

```
    (defrule Posicion-a-la-que-Movernos
        ?id <- (Acto 2)
        (proxMov ?proxDireccion)
        (casilla-actual ?i ?j)
    =>
        (retract ?id)

        (if (eq ?proxDireccion north)
            then (assert (proxCasilla (+ ?i 1) ?j)))
        (if (eq ?proxDireccion south)
            then (assert (proxCasilla (- ?i 1) ?j)))
        (if (eq ?proxDireccion east)
            then (assert (proxCasilla ?i (+ ?j 1))))
        (if (eq ?proxDireccion west)
            then (assert (proxCasilla ?i (- ?j 1))))

        (assert (Acto 3))
```

```
)
```

Esta función permite introducir en la base de hechos un hecho -proxCasilla- para ver cuál sería la casilla a la que nos moveríamos.

2. 2. 2. 4 defrule Comprobar-Casilla-Visitada

```
(defrule Comprobar-Casilla-Visitada
        ?id <- (Acto 3)
        ?id2 <- (proxCasilla ?i ?j)
        ?id3 <- (proxMov ?)
        ?id4 <- (contador ?valor)
        (casilla ?i ?j)
=>
        (retract ?id ?id2 ?id3 ?id4)
        (assert (contador (+ ?valor 1)))
        (assert (Acto 0))
)
```

Esta regla se activa cuando la casilla a la que nos moveríamos sí está visitada, en cuyo caso debemos escoger otra distinta.
En caso de activarse, produce un retroceso en la secuencia de ejecución del programa -hasta Acto 0-.

2. 2. 2. 5 defrule Comprobar-Casilla-NOvisitada

```
(defrule Comprobar-Casilla-NOvisitada
        ?id <- (Acto 3)
        ?id2 <- (proxCasilla ?i ?j)
        ?id3 <- (contador ?valor)
        (not (casilla ?i ?j))
=>
        (retract ?id ?id2)
```

```
        (assert (contador 0))
        (assert (Acto 4))
    )
```

Esta regla se activa cuando la casilla a la que nos moveríamos no está visitada, es decir, la secuencia por defecto continúa.

2. 2. 2. 6 defrule Willy-BLoqueado

```
    (defrule Willy-Bloqueado
        (declare (salience 51))
        ?id <- (contador ?valor)
        ?id2 <- (Acto ?)
        (test (eq ?valor 15))
    =>
        (retract ?id ?id2)
        (assert (contador 0))
        (assert (Acto Especial))
    )
```

Esta función detecta bucles en la secuencia Acto 0 -> Acto 1 -> Acto 2 -> Acto 3 y rompe dicho bucle.

En caso de activarse, rompe la secuencia normal y se introduce el Acto Especial. Es notable destacar que se establece la prioridad a 51 para asegurar su ejecución en el instante correcto.

2. 2. 2. 7 defrule moveWilly-Random

```
    (defrule moveWilly-Random
        ?id <- (Acto Especial)
        (directions $? ?direction $?)
    =>
        (retract ?id)
```

```
        (assert (proxMov ?direction))
        (assert (Acto 4))
    )
```

Función que permite un movimiento en una dirección al azar en caso de bloqueo.

En caso de activarse, continúa la secuencia del programa por el Acto 4.

2. 2. 2. 8 defrule moveWilly-Norte

```
    (defrule moveWilly-Norte
        ?id <- (movimientos ?x)
        ?id2 <- (proxMov ?dire)
        ?id3 <- (Acto 4)
        ?id4 <- (casilla-actual ?i ?j)
        ?id5 <- (ultimoMov ?)
        ?id6 <- (contador ?)

        (test (eq ?dire north))
    =>
        (retract ?id ?id2 ?id3 ?id4 ?id5 ?id6)
        (assert (contador 0))
        (assert (movimientos (+ ?x 1)))
        (assert (ultimoMov north))

        (assert (casilla (+ ?i 1) ?j))
        (assert (casilla-actual (+ ?i 1) ?j))

        (assert (Acto 0))
        (moveWilly ?dire)
```

```
)
```

Función que realiza un movimiento en la dirección norte. Además de eso, actualiza el contador sumándole uno y la casilla-actual con la nueva posición. En caso de que la casilla donde nos movamos ya existiese el assert no modifica ni añade nada porque sería duplicar un hecho. A continuación, volvemos al Acto 0 mediante un assert.

2. 2. 2. 9 defrule moveWilly-Sur

```
(defrule moveWilly-Sur
        ?id <- (movimientos ?x)
        ?id2 <- (proxMov ?dire)
        ?id3 <- (Acto 4)
        ?id4 <- (casilla-actual ?i ?j)
        ?id5 <- (ultimoMov ?)
        ?id6 <- (contador ?)

        (test (eq ?dire south))
=>
        (retract ?id ?id2 ?id3 ?id4 ?id5 ?id6)
        (assert (contador 0))
        (assert (movimientos (+ ?x 1)))
        (assert (ultimoMov south))

        (assert (casilla (- ?i 1) ?j))
        (assert (casilla-actual (- ?i 1) ?j))

        (assert (Acto 0))
        (moveWilly ?dire)
```

```
)
```

Función que realiza un movimiento en la dirección sur. Además de eso, actualiza el contador sumándole uno y la casilla-actual con la nueva posición. En caso de que la casilla donde nos movamos ya existiese el assert no modifica ni añade nada porque sería duplicar un hecho. A continuación, volvemos al Acto 0 mediante un assert.

2. 2. 2. 10 defrule moveWilly-Este

```
(defrule moveWilly-Este
        ?id <- (movimientos ?x)
        ?id2 <- (proxMov ?dire)
        ?id3 <- (Acto 4)
        ?id4 <- (casilla-actual ?i ?j)
        ?id5 <- (ultimoMov ?)
        ?id6 <- (contador ?)

        (test (eq ?dire east))
=>
        (retract ?id ?id2 ?id3 ?id4 ?id5 ?id6)
        (assert (contador 0))
        (assert (movimientos (+ ?x 1)))
        (assert (ultimoMov east))

        (assert (casilla ?i (+ ?j 1)))
        (assert (casilla-actual ?i (+ ?j 1)))

        (assert (Acto 0))
        (moveWilly ?dire)
```

```
)
```

Función que realiza un movimiento en la dirección este. Además de eso, actualiza el contador sumándole uno y la casilla-actual con la nueva posición. En caso de que la casilla donde nos movamos ya existiese el assert no modifica ni añade nada porque sería duplicar un hecho. A continuación, volvemos al Acto 0 mediante un assert.

2. 2. 2. 11 defrule moveWilly-Oeste

```
(defrule moveWilly-Oeste
        ?id <- (movimientos ?x)
        ?id2 <- (proxMov ?dire)
        ?id3 <- (Acto 4)
        ?id4 <- (casilla-actual ?i ?j)
        ?id5 <- (ultimoMov ?)
        ?id6 <- (contador ?)

        (test (eq ?dire west))
=>
        (retract ?id ?id2 ?id3 ?id4 ?id5 ?id6)
        (assert (contador 0))
        (assert (movimientos (+ ?x 1)))
        (assert (ultimoMov west))

        (assert (casilla ?i (- ?j 1)))
        (assert (casilla-actual ?i (- ?j 1)))

        (assert (Acto 0))
        (moveWilly ?dire)
```

```
)
```

Función que realiza un movimiento en la dirección oeste. Además de eso, actualiza el contador sumándole uno y la casilla-actual con la nueva posición. En caso de que la casilla donde nos movamos ya existiese el assert no modifica ni añade nada porque sería duplicar un hecho. A continuación, volvemos al Acto 0 mediante un assert.

2. 2. 2. 12 defrule fireWilly

```
(defrule fireWilly
        (declare (salience 65))
        (hasArrow)
        (or (percepts Sound)
                (percepts Sound ?)
                (percepts ? Sound))
        (directions $? ?direction $?)

        ?id <- (movimientos ?x)
        ?id2 <- (Acto 4)
        ?id3 <- (ultimoMov ?mov)

=>
        (retract ?id ?id2)
        (assert (movimientos (+ ?x 1)))
        (assert (Acto 0))

        (if (eq ?mov north)
                then (fireArrow north))
        (if (eq ?mov south)
```

```
                then (fireArrow south))
            (if (eq ?mov east)
                then (fireArrow east))
            (if (eq ?mov west)
                then (fireArrow west))
    )
```

Función que, en caso de tener la flecha disponible, al dispara hacia la dirección en la que Willy se movió por última vez.
Se establece la prioridad a 65 para asegurar su ejecución el instante correcto.

2. 2. 2. 13 defrule peligroSerpiente

```
    (defrule peligroSerpiente
        (declare (salience 61))
        (or (percepts Sound)
            (percepts Sound ?)
            (percepts ? Sound))

        (not (hasArrow))

        ?id <- (movimientos ?x)
        ?id2 <- (proxMov ?dire)
        ?id3 <- (Acto 4)
        ?id4 <- (casilla-actual ?i ?j)
        ?id5 <- (ultimoMov ?mov)

    =>
        (retract ?id ?id2 ?id3 ?id4 ?id5)
        (assert (Acto 0))
```

```
(assert (movimientos (+ ?x 1)))

(if (eq ?mov north)
        then (moveWilly south))
(if (eq ?mov south)
        then (moveWilly north))
(if (eq ?mov east)
        then (moveWilly west))
(if (eq ?mov west)
        then (moveWilly east))

(if (eq ?mov north)
        then (assert (casilla-actual (+ ?i 1) ?j)))
(if (eq ?mov east)
        then (assert (casilla-actual ?i (+ ?j 1))))
(if (eq ?mov west)
        then (assert (casilla-actual ?i (- ?j 1))))
(if (eq ?mov south)
        then (assert (casilla-actual (- ?i 1) ?j)))

(if (eq ?mov north)
        then (assert (ultimoMov south)))
(if (eq ?mov south)
        then (assert (ultimoMov north)))
(if (eq ?mov east)
        then (assert (ultimoMov west)))
(if (eq ?mov west)
        then (assert (ultimoMov east)))
```

```
)
```

Función que, en caso de no tener la flecha disponible, y si hay una serpiente cerca, realiza un movimiento hacia atrás para esquivar el peligro.

Se establece la prioridad a 61 para asegurar su ejecución el instante correcto.

2. 2. 2. 14 defrule peligroArenas

```
(defrule peligroArenas
        (declare (salience 62))
        (or (percepts Tremor)
                (percepts Tremor ?)
                (percepts ? Tremor))

        ?id <- (movimientos ?x)
        ?id2 <- (proxMov ?dire)
        ?id3 <- (Acto 4)
        ?id4 <- (casilla-actual ?i ?j)
        ?id5 <- (ultimoMov ?mov)

=>
        (retract ?id ?id2 ?id3 ?id4 ?id5)
        (assert (Acto 0))
        (assert (movimientos (+ ?x 1)))

        (if (eq ?mov north)
                then (moveWilly south))
        (if (eq ?mov south)
                then (moveWilly north))
```

```
        (if (eq ?mov east)
            then (moveWilly west))
        (if (eq ?mov west)
            then (moveWilly east))

        (if (eq ?mov north)
            then (assert (casilla-actual (+ ?i 1) ?j)))
        (if (eq ?mov east)
            then (assert (casilla-actual ?i (+ ?j 1))))
        (if (eq ?mov west)
            then (assert (casilla-actual ?i (- ?j 1))))
        (if (eq ?mov south)
            then (assert (casilla-actual (- ?i 1) ?j)))

        (if (eq ?mov north)
            then (assert (ultimoMov south)))
        (if (eq ?mov south)
            then (assert (ultimoMov north)))
        (if (eq ?mov east)
            then (assert (ultimoMov west)))
        (if (eq ?mov west)
            then (assert (ultimoMov east)))
    )
```

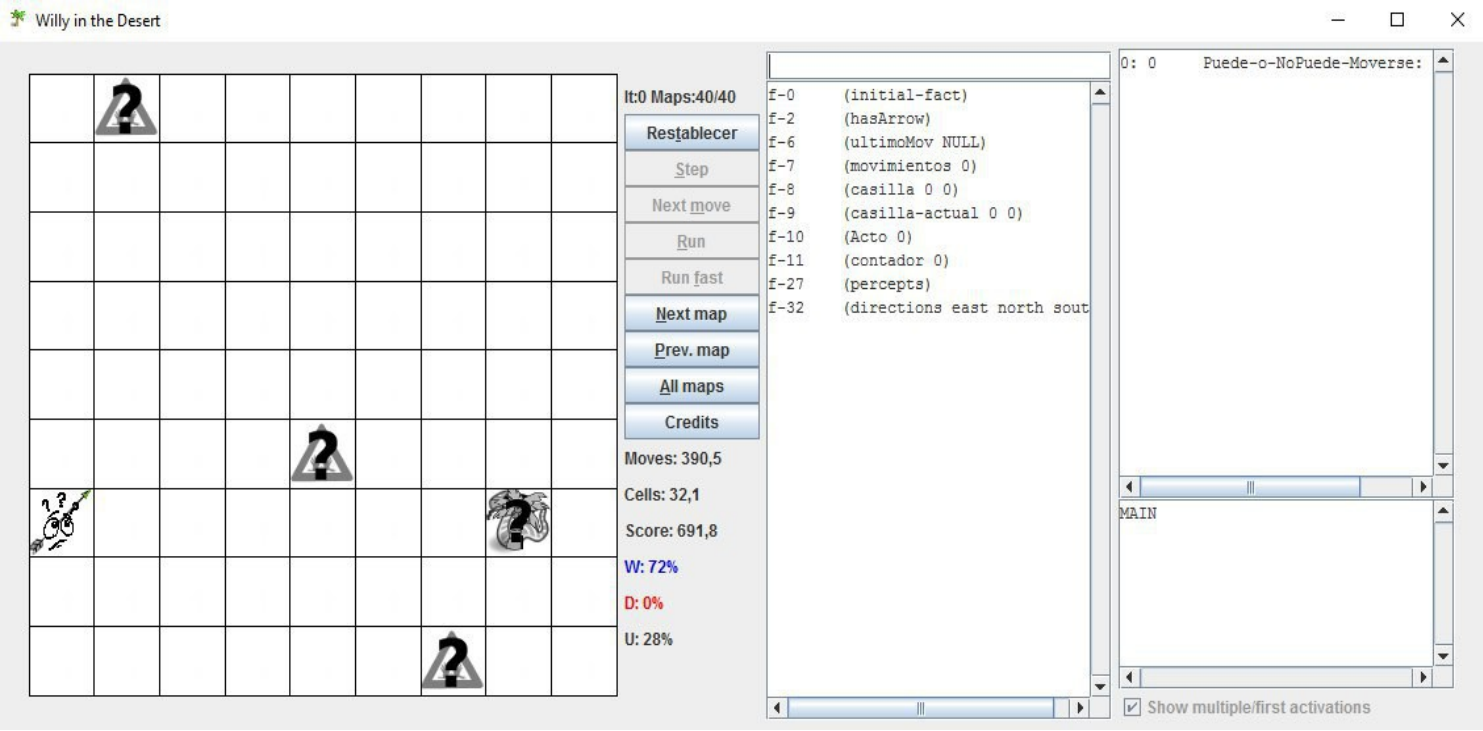
Función que, en caso de que haya unas arenas movedizas cerca, realiza un movimiento hacia atrás para esquivar el peligro.
Se establece la prioridad a 62 para asegurar su ejecución el instante correcto.

CAPÍTULO 3

Resultados

3.1 Resultados

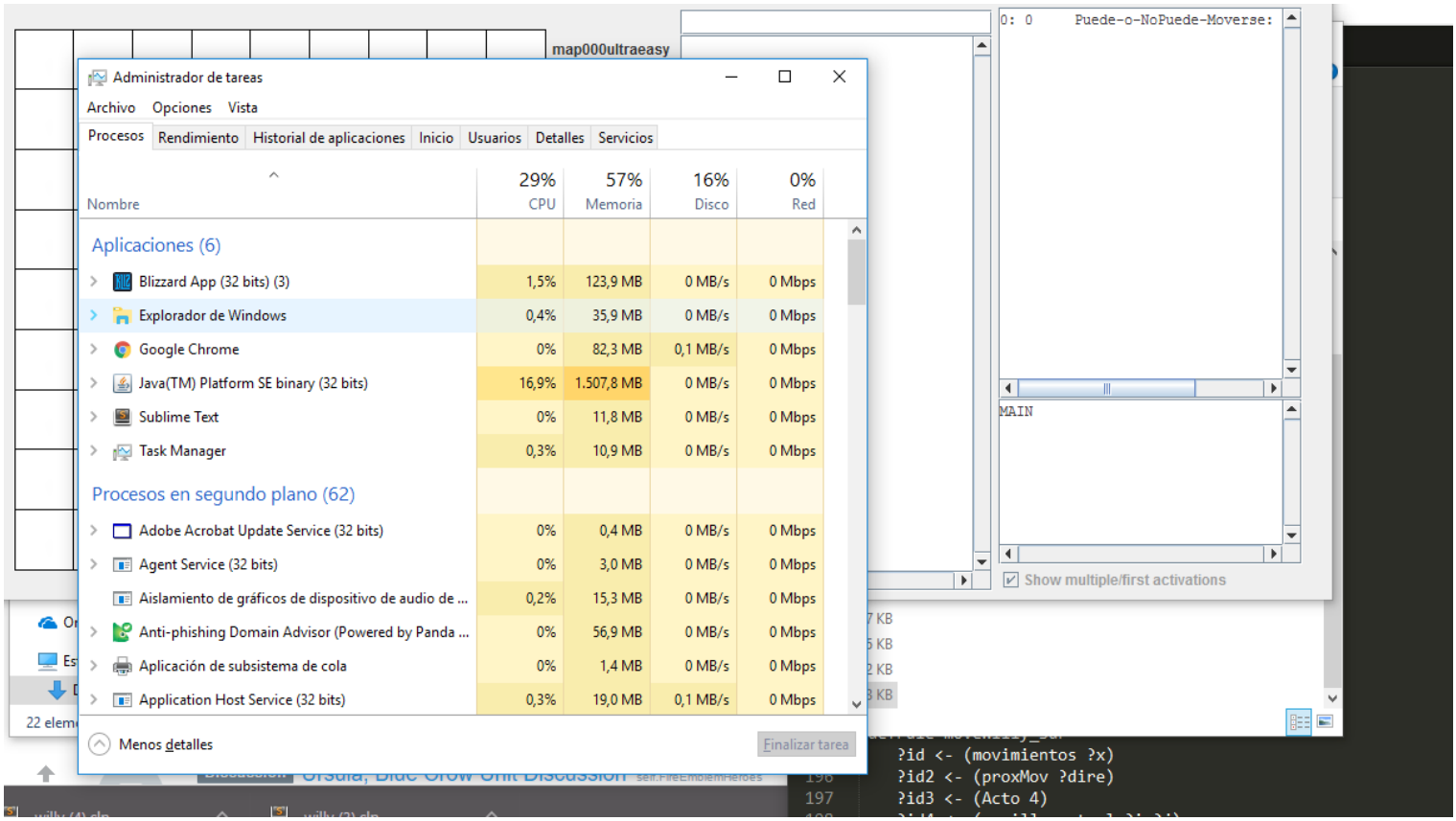
Tras realizar el modo 'all maps' del ejecutable los resultados obtenidos son estos:



3.2 Consideraciones

Existía una versión anterior a la descrita y enviada en este trabajo que, a nuestro juicio, estaba mejor pensada pero que sobrecargaba el sistema hasta el punto de cierres inesperados al ejecutar 'all maps' y tiempos de espera en el modo 'run fast' que hacía inviable seguir desarrollando algo así. A continuación se adjunta una captura donde se ve

cómo la RAM consumida era de 1'5 gb (como curiosidad, el videojuego Diablo 3 a 180 fps consume 2'2 gb de RAM, sólo 0'7 gb menos en algo bastante más exigente).



Las funciones eliminadas son las siguientes:

```
(defrule Willy-Bloqueado-CENTRO
        (declare (salience 51))
        (casilla ?i1 ?j1) ;north
        (casilla ?i2 ?j2) ;south
        (casilla ?i3 ?j3) ;east
        (casilla ?i4 ?j4) ;west
```

```
        (casilla-actual ?x ?y)
        (directions ? ? ? ?)

        (test (and (eq (+ ?x 1) ?i1) (eq ?j1 ?y))) ;north
        (test (and (eq (- ?x 1) ?i2) (eq ?j2 ?y))) ;south
        (test (and (eq ?x ?i3) (eq ?j3 (+ ?y 1)))) ;east
        (test (and (eq ?x ?i4) (eq ?j4 (- ?y 1)))) ;west
=>
        (assert (bloqueado))
)

(defrule Willy-Bloqueado-ESQUINA
        (declare (salience 51))
        (casilla ?i1 ?j1) ;north
        (casilla ?i2 ?j2) ;south
        (casilla-actual ?x ?y)
        (directions ? ?)

        (or
                (and
                        (test (and (eq (- ?x 1) ?i1) (eq ?y ?j1)))
                        (test (and (eq ?x ?i2) (eq (+ ?y 1) ?j2)))
                ) ;esquina 0-0

                (and
                        (test (and (eq (- ?x 1) ?i1) (eq ?y ?j1)))
                        (test (and (eq ?x ?i2) (eq (- ?y 1) ?j2)))
                ) ;esquina 0-8
```

```
                (and
                        (test (and (eq (+ ?x 1) ?i1) (eq ?y ?j1)))
                        (test (and (eq ?x ?i2) (eq (+ ?y 1) ?j2)))
                ) ;esquina -8-0

                (and
                        (test (and (eq (+ ?x 1) ?i1) (eq ?y ?j1)))
                        (test (and (eq ?x ?i2) (eq (- ?y 1) ?j2)))
                ) ;esquina -8--8
        )


=>
        (assert (bloqueado))
)

(defrule Willy-Bloqueado-PARED
        (declare (salience 51))
        (casilla ?i1 ?j1) ;north
        (casilla ?i2 ?j2) ;south
        (casilla ?i3 ?j3)
        (casilla-actual ?x ?y)
        (directions ? ? ?)

        (or
                (and
                        (test (and (eq (+ ?x 1) ?i1) (eq ?j1 ?y)))
```

```
                    (test (and (eq ?x ?i2) (eq ?j2 (+ ?y 1))))
                    (test (and (eq (- ?x 1) ?i3) (eq ?j3 ?y)))
                ) ;pared west

            (and
                (test (and (eq (+ ?x 1) ?i1) (eq ?j1 ?y)))
                (test (and (eq ?x ?i2) (eq ?j2 (- ?y 1))))
                (test (and (eq (- ?x 1) ?i3) (eq ?j3 ?y)))
            ) ;pared east

            (and
                (test (and (eq ?x ?i1) (eq ?j1 (- ?y 1))))
                (test (and (eq ?x ?i2) (eq ?j2 (+ ?y 1))))
                (test (and (eq (- ?x 1) ?i3) (eq ?j3 ?y)))
            ) ;pared north

            (and
                (test (and (eq ?x ?i1) (eq ?j1 (- ?y 1))))
                (test (and (eq ?x ?i2) (eq ?j2 (+ ?y 1))))
                (test (and (eq (+ ?x 1) ?i3) (eq ?j3 ?y)))
            ) ;pared south
        )
    =>
        (assert (bloqueado))
    )
```

NOTA: los (declare (salience 51)) fueron añadidos en un último intento de reconducir por el buen camino a estos monstruos devora-RAM.