Atomic Player is < X, Y, S, δint, δext, λ, ta > where

params is

est: N; // Estrategia a utilizar

end params

X is

in: N x $\mathbb{R}_0^+$ x $\mathbb{R}_0^+$; // (nro de evento, peso reducido, distancia recorrida)

end X

Y is

out: $\mathbb{R}_0^+$; // El peso del objeto saliente.

end Y

S is

// (Lista de pesos y tiempo de interarribo ordenada según la estrategia, estrategia)

s: List ($\mathbb{R}_0^+$× Time) × N;

end S

δint(xs) is

s = tail xs;

end δint

δext(xs, e, (in, x))) is

s = PlayerLibrary.calcularEstado(xs, x, est, e)

end δext

λ(xs) is

out = (head xs).fst();

end λ

ta(xs) is

(head xs).snd();

end ta

end Atomic

Estado inicial: xs: lista generada y ordenada según la estrategia.

est: la estrategia que es pasada por parámetro

(0 = PC, 1..4 = estrategias de jugador)

```
functions PlayerLibrary is
    function calularEstado is
        xs: List (ℝ₀⁺ × Time),
        pesoGanador: ℝ₀⁺,
        est: N,                 // estrategia
        e: Time                 // tiempo transcurrido
        → res: List (ℝ₀⁺ × Time) × N;
        defwhere
            defcases
                if (est == 3) ⇒ res = mayorQue(xs, pesoGanador)
                if (est == 4) ⇒ res = sacarPrimerElemento ⌢ tail(xs)
                default ⇒ res = defaultHead ⌢ tail(xs)
            end defcases
        where
            sacarPrimerElemento = head(xs).fst()
            defaultHead = (head(xs).fst(), head(xs).snd() – e)
        endefwhere
    end function

    function mayorQue is
        xs: List (ℝ₀⁺ × Time), n: ℝ₀⁺ → res: List (ℝ₀⁺ × Time);
        b: Bool;
        b = false;
        foreach x in xs
            if (x.fst() > n ^ b = false ) ⇒
                res = (x.fst(), 0) ⌢ res;
                b = true;
            else
                res = res ⌢ x;
        end foreach
    end function
end functions
```

Atomic Cinta is < X, Y, S, δint, δext, λ, ta > where

params is

Vc = 1; //Velocidad constante

longitud = 25; //Longitud de la cinta

end params

X is

// Pesos de los objetos.

in_human: $\mathbb{R}_0^+$; //Puerto de entrada para Player

in_pc: $\mathbb{R}_0^+$; //Puerto de entrada para PC

end X

Y is

// El peso del objeto ganador de una colisión.

out_pesoVencedor: $\mathbb{R}_0^+$;

// Llego un objeto al extremo del jugador o de la PC

out_llegadas: { llego_jugador, llego_pc}

// Información sobre una colisión (información del ganador, peso con el que continua el objeto, distancia de la colisión)

out_colision: {gano_pc, gano_player, empate} × $\mathbb{R}_0^+$ × $\mathbb{R}_0^+$

end Y

S is

// Lista de pesos y distancias recorridas del jugador y de la PC, y tiempo del siguiente evento (colision o llegada)

s: List ($\mathbb{R}_0^+$× $\mathbb{R}_0^+$) × List ($\mathbb{R}_0^+$× $\mathbb{R}_0^+$) × Time;

end S

δint((xs, ys, σ) is

defwhere

defcases

if ((head actual_xs).snd() == longitud) ⇒

case s = (tail actual_xs, actual_ys, nextEvent(tail actual_xs, actual_ys))

if ((head actual_ys).snd() == longitud) ⇒

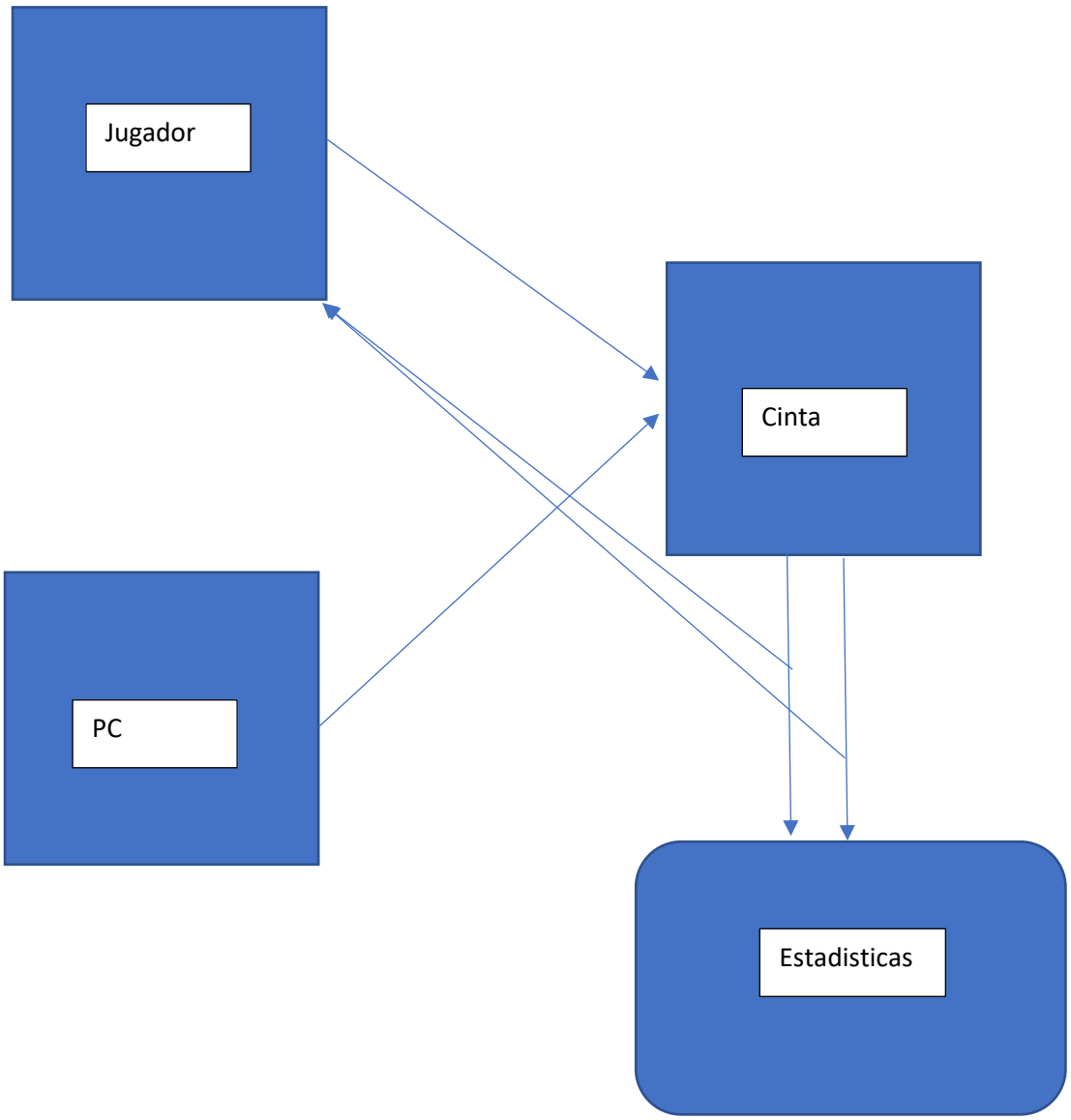
case s = (actual_xs, tail actual_ys, nextEvent(actual_xs, tail actual_ys))

```
    if (caja_jugador == caja_pc) ⇒
        case s = (tail actual_xs, tail actual_ys, nextEvent(tail actual_xs, tail actual_ys))
    if (caja_jugador < caja_pc) ⇒
        case s = (tail actual_xs, ganador_ys, nextEvent(tail actual_xs, ganador_ys))
    if (caja_jugador > caja_pc) ⇒
        case s = (ganador_xs, tail actual_ys, nextEvent(ganador_xs, tail actual_ys))
    end defcases
    where
        actual_xs = actualizarDistancias(xs, σ);
        actual_ys = actualizarDistancias(ys, σ);
        caja_jugador = (head actual_xs).fst() * (head actual_xs).snd()
        caja_pc = (head actual_ys).fst() * (head actual_ys).snd()
        ganador_ys = (newPeso(head actual_ys, head actual_xs),
                        head actual_ys.snd()) ⌒ tail actual_ys)
        ganador_xs = (newPeso(head actual_xs, head actual_ys),
                        head actual_xs.snd()) ⌒ tail actual_xs)
    end defwhere
end δint

δext((xs, ys, σ), e, (in, x)) is
    defwhere
        defcases
            if (in == in_human) ⇒
                case s = (actual_xs ⌒ (x, 0), actual_ys, σ - e);
            if (in == in_pc) ⇒
                case s = (actual_xs, actual_ys ⌒ (x,0), σ - e);
        end defcases
    where
        actual_xs = actualizarDistancias(xs, e);
        actual_ys = actualizarDistancias(ys, e);
    end defwhere
end δext


λ((xs, ys, σ)), is
    defwhere
        defcases
```

```
            if (distancia_jugador == longitud) ⇒
                case out_llegadas = llego_jugador
            if (distancia_pc == longitud) ⇒
                case out_llegadas = llego_pc
            if (caja_jugador == caja_pc) ⇒
                case out_colision = (empate, 0, (head xs).snd())
            if (caja_jugador < caja_pc) ⇒
                case out_colision = (gano_pc, newPeso(head ys, head xs),
                                     (head xs).snd());
            if (caja_jugador > caja_pc) ⇒
                case out_colision = (gano_jugador, calcPeso(head xs, head ys),
                                     (head xs).snd())
        end defcases
    where
        distancia_jugador = nuevaDistancia((head xs).snd(), σ)
        distancia_pc = nuevaDistancia((head ys).snd(), σ)
        caja_jugador = (head xs).fst() * distancia_jugador
        caja_pc = (head ys).fst() * distancia_jugador
    end defwhere
end λ

ta((xs, ys, σ)) is
    σ;
end ta

end Atomic
```

Estado inicial: Las dos listas vacias, y sigma infinito.

```
functions CintaLibrary is
    //Actualiza la distancia recorrida de todos elementos de la lista.
    function actualizarDistancias is
        S: List (R × R), tiempoTrans: R → res: List (R × R);
        foreach x in S
            x.snd() = x.snd() + tiempoTrans * Vc;
        end foreach
        res = S;
    end function

    //Retorna el nuevo peso del objeto vencedor con la funcion indicada en el
proyecto.
    function newPeso is
        vencedor, perdedor: R × R → res: R;
        res = vencedor.fst() *
            ((perdedor.fst() * perdedor.snd()) / (vencedor.fst() * obj_vencedor.snd()))
    end function


    //Devuelve el tiempo que necesita transcurrir para que ocurra el siguiente
evento(colision/llegada).
    function nextEvent is
        xs, ys: List (R × R) → res: R;
        defcases
            if ( xs = {} ) ⇒ case res = (longitud - head ys.snd()) / Vc ;
            if ( ys = {} ) ⇒ case res = (longitud - head xs.snd()) / Vc ;
            if ( xs != {} ^ ys != {} ) ⇒ case res = ((longitud - head xs.snd() - head
ys.snd()) / 2) / Vc ;
            if ( xs = {} ^ ys = {} ) ⇒ case res =  ∞
        end defcases
    end function
end functions
```


Jugador
Cinta
PC
Estadisticas