



Universidad
Carlos III de Madrid

Grado en Ingeniería Informática

Curso 2020/2021

Ingeniería del Conocimiento

Práctica 1: Sistemas de Producción

Autores:

Alba Reinders Sánchez 100383444
Alejandro Valverde Mahou 100383383

Grupo 83 Leganés

Índice

1. Introducción	3
2. Manual técnico	4
2.1. Ontología	5
2.2. Reglas	6
2.2.1. Reglas genéricas	6
2.2.2. Reglas Twister	7
2.2.3. Reglas Tres en Raya	7
3. Manual de usuario	8
4. Pruebas realizadas	9
4.0.1. Prueba 1-1: Comportamiento normal en el Tres en Raya con victoria del robot	9
4.0.2. Prueba 1-2: Comportamiento normal en el Tres en Raya con victoria del paciente	9
4.0.3. Prueba 1-3: Comportamiento normal en el Tres en Raya con empate	9
4.0.4. Prueba 2-1: Comportamiento normal en el Twister con victoria del paciente	9
4.0.5. Prueba 2-2: Comportamiento normal en el Twister con caída del paciente	9
4.0.6. Prueba 3-1: Comportamiento despistado repetir explicación en el Twister	9
4.0.7. Prueba 3-2: Comportamiento despistado repetir movimiento en el Twister con finalización normal	10
4.0.8. Prueba 3-3: Comportamiento despistado repetir movimiento en el Twister con finalización por demasiados despistes	10
4.0.9. Prueba 4-1: Comportamiento despistado repetir movimiento en el Tres en Raya con finalización normal	10
4.0.10. Prueba 4-2: Comportamiento despistado repetir movimiento en el Tres en Raya con finalización por demasiados despistes	10
4.0.11. Prueba 5-1: Comportamiento enérgico repetir el juego	10
4.0.12. Prueba 5-2: Comportamiento enérgico impedir colocar fuera de turno en el Tres en Raya con finalización normal	10
4.0.13. Prueba 5-2-bis: Comportamiento enérgico impedir colocar fuera de turno en el Tres en Raya con finalización por demasiadas trampas	10
5. Conclusiones	12
6. Comentarios personales	12

1. Introducción

En esta primera práctica se aborda la implementación de un **sistema de producción (SP)** en CLIPS que lleve a cabo la ejecución de una sesión de un especialista con un paciente en la que interactúan un humano y un robot NAO, donde este último adopta el papel de especialista.

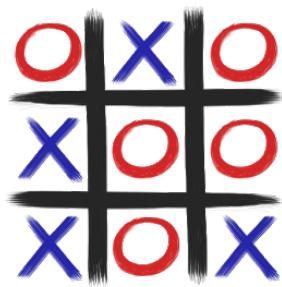


Robot NAO

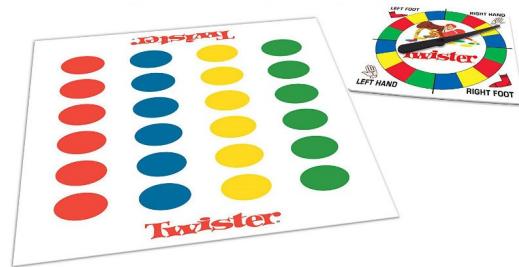
En concreto, se crea un *SP* con toda la información respecto a los personajes y el entorno en el que se lleva a cabo la interacción, así como posibles desviaciones durante la sesión en las que el robot debe reaccionar y modificar sus acciones en consecuencia.

La interacción entre el paciente y el robot se realiza a través del desarrollo de 2 juegos: el **Twister** y el **Tres en raya**. Además, el sistema está adecuado para tratar con pacientes que puedan presentar 2 personalidades distintas, aparte del comportamiento base, **despistado** y **energético**.

Se han elegido estos juegos por ser ejemplos de actividades sencillas, que requieren de un número reducido de reglas, para demostrar las aplicaciones de los *SP*. Por el mismo motivo, se han elegido únicamente 2 posibles comportamientos básicos. Si se deseara aplicar sobre un problema real no serían suficientes, y sería necesario tener en cuenta un mayor número de personalidades.



Juego del Tres en Raya



Juego del Twister

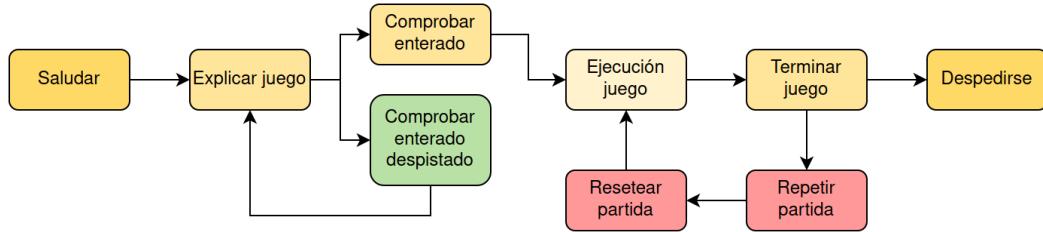
El documento consiste en el manual técnico con la descripción de la implementación, el manual de usuario con la explicación de cómo usar el programa, las pruebas realizadas y el análisis de los resultados, y para finalizar una serie de conclusiones y comentarios personales.

2. Manual técnico

El SP está compuesto por la **ontología** y las **reglas**. A continuación, se explican ambas justificando todas las decisiones que se han tomado durante la implementación.

Los flujos de acciones que se han planteado para resolver los problemas son los siguientes, donde los estados de color rojo solo se pueden realizar si el *paciente* es *enérgico* y los de color verde si el *paciente* es *despistado*.

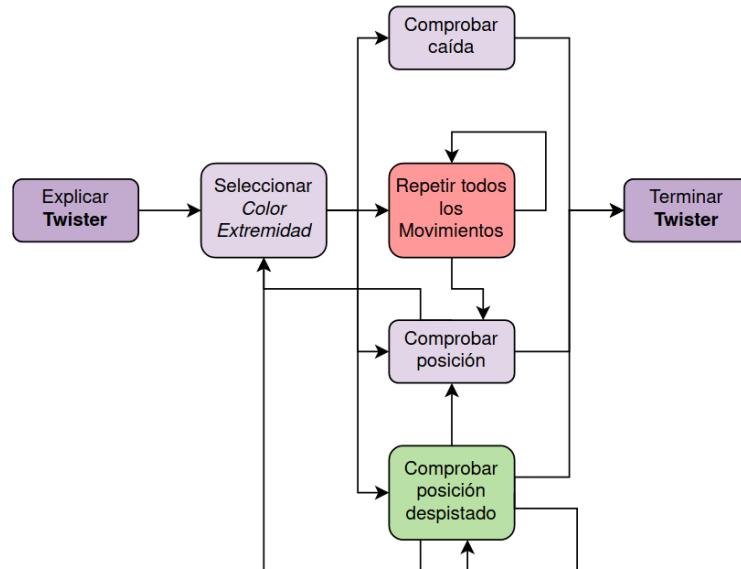
El flujo de una sesión consiste en: saludar al *paciente* y explicarle el juego. Si se comprueba que el *paciente* no lo ha entendido correctamente, se repite la explicación. Una vez el *paciente* lo ha entendido, se procede a ejecutar el juego. Cuando finaliza el mismo, se comprueba si el *paciente* tiene energías para seguir jugando. Si es así, se repite el juego. En caso contrario, se termina la sesión.



Flujo genérico de una sesión

Si el juego elegido es el **Twister**, el flujo consiste en, una vez explicado y entendido el juego, seleccionar un color y extremidad que el *paciente* tendrá que realizar. Si el *paciente* la realiza correctamente, se pasará a elegir otro color y extremidad. Una vez se haga correctamente 5 veces, el juego terminará. Por otro lado, si el *paciente* se cae, el juego terminará también.

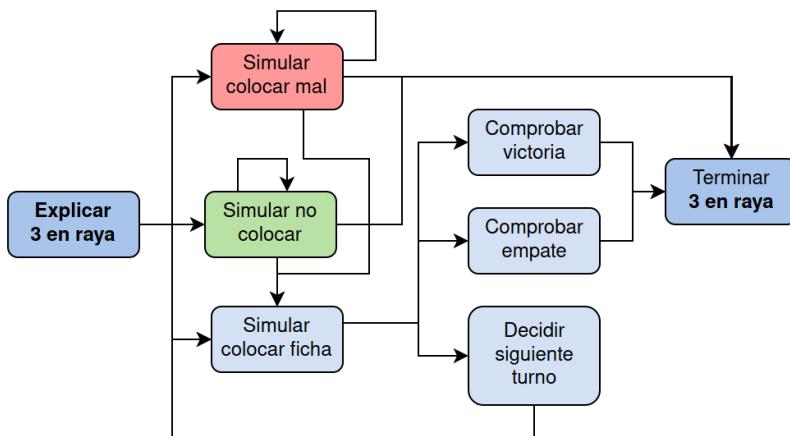
Además, si el *paciente* está despistado, se le repetirá el último comando y, en caso de que se lo tenga que repetir más de 6 veces, se considera que el *paciente* está demasiado despistado para continuar y se termina la sesión. Por último, si el *paciente* cambia su posición, se le repetirán todas las órdenes, para que pueda volver a colocarse correctamente.



Flujo del Twister

Si el juego es el **Tres en Raya**, el flujo consiste en, una vez explicado y entendido el juego, simular colocar una ficha por turnos, pasando del *paciente* al *robot* (empieza siempre el *paciente*). Si el *paciente* está despistado y no coloca ficha se le recuerda un máximo de 3 veces seguidas, y en caso de que siga despistado, se terminará la sesión. Por otro lado, si se despista un total de 6 veces, la sesión terminará, por el mismo motivo.

Si por el contrario tiene muchas energías y coloca ficha cuando no es su turno, se le regaña, y se le impedirá colocar la ficha. Además, si intenta colocar ficha fuera de turno más de 3 veces, se terminará la partida, y se le dará la victoria al *robot*. Siempre que no haya tres en raya o un empate (todas las casillas ocupadas sin tres en raya) se seguirá el flujo del juego.



2.1. Ontología

La ontología que se plantea para la solución está compuesta de las siguientes clases.

- **JUEGO** → Representa el juego que se va a llevar a cabo a lo largo de la sesión. Contiene, además del nombre (*id*) del juego, la explicación del mismo, junto con la indicación de quién debe empezar (*paciente* o *robot*). Esto se ha decidido porque para cada juego comienza uno de los dos:

En el caso del **Twister**, el *robot* será siempre el que empiece la partida, dado que será el *paciente* el que siempre 'juegue', mientras que el *robot* elige las acciones.

Por otro lado, se ha decidido que para el caso del **Tres en Raya** siempre empiece el paciente. Esto se ha hecho porque, en este juego, el jugador que comienza tiene ventaja y, dado que el juego forma parte de una terapia, se quiere potenciar que el *paciente* gane con mayor probabilidad.

- **CONTROL** → Esta clase tendrá una única instancia, y se usa para localizar las distintas operaciones que controlan el sistema en una sola instancia. En el caso excepcional de que se repita la partida (comportamiento que solo puede darse con *pacientes* de personalidad *enérgica*) esta instancia se borra y se reinicia.

Esta clase permite controlar de quién es el turno, el número de ciclos, junto al número máximo de ciclos, el número de fallos, el juego que se lleva a cabo en la sesión, y el número de veces que se repite la partida.

- **PACIENTE** → Representa al paciente que está en la sesión. Se ha decidido usar una clase en lugar de un *fact* para representarlo porque, además de la personalidad del paciente, también almacena el nombre del mismo. Esto se hace para poder enviar mensajes personalizados, y que el paciente se encuentre en un ambiente más cómodo e informal.

- **COLOR** → Representa cada uno de los cuatro colores posibles para el juego del **Twister**. Puede ser *rojo*, *verde*, *amarillo* o *azul*.

- **EXTREMIDAD** → Representa cada una de las cuatro extremidades posibles para el juego del **Twister**. Puede ser *pie derecho, pie izquierdo, mano derecha o mano izquierda*.
- **ELECCION** → Representa cada una de las elecciones o comandos que hace el *robot* y que tiene que realizar el *paciente* en el juego del **Twister**. Además del color y la extremidad elegidas, para poder realizar comprobaciones adicionales, se incluye el número de veces que se ha tenido que repetir una misma elección, y el orden por el que se han dicho.
- **CASILLA** → Representa cada una de las 9 casillas del juego del **Tres en Raya**. Además de sus coordenadas, se tiene su valor, que puede ser *vacío* si no se ha colocado nada todavía, o *paciente* o *robot*, en función de quién coloque la ficha.

2.2. Reglas

Según se ha planteado el *SP*, en la reglas reside toda la funcionalidad del sistema. Por ello, el número de reglas creadas es bastante grande. Sin embargo, considerando todas las distintas variaciones e interacciones que se han tenido en cuenta, resulta lógico que se necesiten más reglas.

Esta cantidad de reglas permite que los datos que se tienen que introducir por un usuario del sistema sea mínima, y hace que todo el flujo sea controlado por las reglas.

También destaca que, como se ha enfocado a dos personalidades concretas, hay reglas que son exclusivas para estas personalidades. Esto hace que al estar enfocado a una solución muy concreta, la resolución es mucho más detallada y potente. Pero a cambio se pierde capacidad de generalización a la hora de añadir nuevas personalidades.

Si se quisiera añadir un personalidad nueva habría que reescribir y modificar reglas, así como crear nuevas. A pesar de esto, se ha decidido esta forma para poder realizar un desarrollo de la sesión lo más completa posible.

2.2.1. Reglas genéricas

Estas reglas son las que sirven para ambos juegos y donde da igual la personalidad del paciente:

- **Saludar**: el *robot* saluda de manera personalizada al *paciente* por su nombre.
- **Explicar**: el *robot* explica al *paciente* el juego que se haya elegido.
- **Comprobar-enterado**: el *robot* comprueba que el *paciente* se haya enterado de la explicación del juego.
- **Despedirse**: el *robot* se despide de manera personalizada del *paciente* por su nombre.

Si el *paciente* tiene de personalidad *despistado*, existe la regla **Comprobar-enterado-despistado** para repetir la explicación del juego y la regla **Demasiadas-equivocaciones-despistado** para terminar la sesión porque se ha equivocado más de 6 veces y por tanto se considera que el *paciente* puede no estar muy atento para continuar con el juego.

Se añade la regla **Repetir-partida** para volver a jugar otra partida. Esto sucede exclusivamente si el *paciente* es *enérgico*, ya que se considera que puede que tenga más energías para seguir jugando.

Por lo tanto, una vez que se acaba una partida se activa esta regla y, dependiendo del juego, las siguientes: **Resetear-partida-tres-en-rayo** y **Resetear-partida-twister**. Estas sirven para reiniciar las variables del juego.

Además, se crea la regla **Auxiliar-fin-reseteo**, que lo que hace es finalizar el proceso de reinicio de variables.

Por último, se tiene la regla **Demasiadas-equivocaciones-energico**, la cual tiene como fin terminar la sesión si el *paciente* intenta colocar un ficha cuando no es su turno más de 3 veces. Este comportamiento es exclusivo del *enérgico*, ya que se considera que ha hecho demasiadas trampas para una sola partida.

2.2.2. Reglas Twister

En cuanto a las reglas exclusivas para el **Twister**, para un comportamiento normal, se crean las siguientes:

- **Robot-selecciona**: el *robot* selecciona de forma aleatoria un color y una extremidad para que el *paciente* realice la acción.
- **Comprobar-posicion**: el *robot* comprueba que el *paciente* ha realizado correctamente la acción que le ha dicho y aumenta el contador de ciclos.
- **Comprobar-caida**: el *robot* comprueba que el *paciente* se ha caído y que por tanto se acaba el juego.
- **Terminar-juego**: por otro lado, si el contador de ciclos alcanza los ciclos máximos, se termina el juego.

Según la personalidad del paciente se pueden activar además las siguientes reglas:

Si es *despistado*:

- **Comprobar-posicion-despistado**: si el *paciente* es *despistado* se puede activar esta regla con el objetivo de que el *robot* le repita la acción al *paciente* porque no la está haciendo.
- **Saltar-ciclo**: cuando el *robot* ha tenido que repetir la misma acción más de 2 veces porque el *paciente* no la realiza se procede a pasar al siguiente movimiento.

Si es *enérgico*:

- **Comprobar-posicion-energico**: si el *paciente* es *enérgico* se puede activar esta regla para simular que el *paciente* se ha movido de su posición porque no se esté quieto y por tanto el *robot* le avisa y le repite todos los movimientos.
- **Repetir-orden**: regla para repetir todos los movimientos en orden gracias al orden que acompaña a cada elección.
- **Auxiliar-repetir-orden**: sirve para dar por finalizado la repetición de los movimientos.

2.2.3. Reglas Tres en Raya

Las reglas creadas para un comportamiento normal del **Tres en Raya** son:

- **Simular-colocar**: sirve para representar que uno de los jugadores ha colocado una ficha en una casilla vacía.
- **Comprobar-victoria**: engloba a **Comprobar-victoria-h**, **Comprobar-victoria-v**, **Comprobar-victoria-d1** y **Comprobar-victoria-d2**. Comprueban si, tras colocar una ficha, alguno de los dos jugadores gana la partida. Tiene que dividirse en varias reglas porque existen estas 4 posibles condiciones de victoria (horizontal, vertical, y las dos diagonales). Estas reglas tienen que tener mayor prioridad que la regla de empate, para que el sistema no pueda elegir empate cuando se cumple victoria.
- **Comprobar-empate**: se comprueba si se han rellenado las 9 casillas, y no ha habido victoria de ningún jugador.
- **Decidir-siguiente-robot**: como el *paciente* siempre comienza la partida, será el turno del *robot* siempre que haya colocadas un número impar de fichas.
- **Decidir-siguiente-paciente**: por el mismo motivo que la regla anterior, será el turno del *paciente* siempre que haya colocadas un número par de fichas.

Para cuando el *paciente* es *enérgico* se tiene la regla **Simular-colocar-mal** que simula cuando el *paciente* intenta colocar una ficha sin ser su turno porque tiene mucha energía y por tanto no puede esperar a su turno. La regla lo que hace es sumar uno al contador de fallos y decirle que no puede colocar porque no es su turno.

Si el *paciente* es *despistado* se tiene la regla **Simular-no-colocar** que simula cuando el *paciente* no coloca una ficha cuando es su turno porque no está muy atento y se le olvida que es su turno. La regla lo que hace es sumar uno al contador de fallos y recordarle que es su turno.

3. Manual de usuario

El programa realizado es muy sencillo de cara a los parámetros que tiene que tener en cuenta el usuario a la hora de utilizarlo.

En *deffacts init* debe escribir dentro del *fact 'juego'* el nombre de aquel que se va a realizar: **tres-en-rayo** ó **twister**.

En *definstances init* debe especificar siempre el nombre del paciente en la *instancia* de la clase PACIENTE, escribiéndolo en el *slot 'nombre'*. Además, si el paciente tiene la personalidad de enérgico o despistado se tiene que escribir en el *slot 'personalidad'*: **energico** ó **despistado**.

Si no se pone ninguna personalidad ó si se pone una que no sea ninguna de las anteriores, el sistema realizará el comportamiento por defecto sin variaciones.

Para un caso de uso donde se quiere llevar a cabo una sesión con el juego del **Tres en Raya** y con un paciente con personalidad *despistado*, se deben realizar los siguientes pasos:

- Paso 1: escribir **tres-en-rayo** en el *fact 'juego'* de *deffacts init*.
- Paso 2: escribir, dentro de la *instancia* de la clase PACIENTE de *definstances init*, el nombre del paciente (por ejemplo Pepe) en el *slot 'nombre'*. Y en el *slot 'personalidad'* escribir **despistado**.

A continuación se muestra el ejemplo de código de este caso de uso:

```
(deffacts init
  (juego tres-en-rayo)
)
(definstances init
  ([pepe] of PACIENTE (nombre Pepe) (personalidad despistado))
)
```

Respecto a la salida del programa, se va mostrando a través de la terminal el flujo de la sesión, tanto lo que le dice el robot al paciente como cambios en el estado del juego según los parámetros introducidos.

4. Pruebas realizadas

Nota: varios archivos pueden ser reutilizados para diferentes pruebas dado el carácter aleatorio de cada ejecución. Por el mismo motivo, el ejecutar una prueba concreta no quiere decir que se obtendrá el mismo resultado, sino que es uno de los resultados posibles.

4.0.1. Prueba 1-1: Comportamiento normal en el Tres en Raya con victoria del robot

Archivo de entrada: **prueba-1.clp**

Archivo de salida: **salida-prueba-1-1.txt**

Esta prueba consiste en evaluar si el sistema genera el comportamiento esperado. Concretamente se evalúa que todo funcione para el caso en que la ejecución no sufre alteraciones en el juego del **Tres en Raya**, y el robot consigue una victoria (Casillas 1-1; 1-2; 1-3).

4.0.2. Prueba 1-2: Comportamiento normal en el Tres en Raya con victoria del paciente

Archivo de entrada: **prueba-1.clp**

Archivo de salida: **salida-prueba-1-2.txt**

Esta prueba consiste, al igual que la anterior, en evaluar si el sistema genera el comportamiento esperado para el caso en que la ejecución no sufre alteraciones en el juego del **Tres en Raya**, y, en este caso, el paciente consigue una victoria (Casillas 2-1; 2-2; 2-3 ó 3-1; 2-2; 1-3).

4.0.3. Prueba 1-3: Comportamiento normal en el Tres en Raya con empate

Archivo de entrada: **prueba-1.clp**

Archivo de salida: **salida-prueba-1-3.txt**

Esta prueba es similar a las anteriores, pero en este caso se comprueba que el empate funciona correctamente.

4.0.4. Prueba 2-1: Comportamiento normal en el Twister con victoria del paciente

Archivo de entrada: **prueba-2.clp**

Archivo de salida: **salida-prueba-2-1.txt**

En esta prueba se comprueba que el paciente cumpla una sesión con normalidad en el juego del **Twister**, y que no se caiga, ganando así el juego. La secuencia de acciones que realiza el paciente es: *pie izquierdo en verde; pie izquierdo en amarillo; pie izquierdo en verde; pie derecho en rojo; pie izquierdo en verde*.

Nota: a pesar de que parece que la secuencia es muy sencilla, pues repite 4 veces una instrucción para el pie izquierdo, este es un comportamiento que puede darse a la hora de jugar al Twister en la realidad, por lo que no se ha decidido realizar ningún tipo de control sobre las elecciones para evitar estas repeticiones.

4.0.5. Prueba 2-2: Comportamiento normal en el Twister con caída del paciente

Archivo de entrada: **prueba-2.clp**

Archivo de salida: **salida-prueba-2-2.txt**

En este caso se comprueba que el paciente no llegue a vencer el juego porque se caiga antes. Esto solo puede ocurrir una vez se han realizado al menos 3 acciones, ya que se ha considerado que un paciente no puede caerse antes de haber hecho estos 3 movimientos. Las acciones que ha realizado el paciente antes de caerse han sido: *mano derecha en azul; pie derecho en verde; pie izquierdo en verde; pie derecho en verde*. Al ir a hacer la acción *pie derecho en amarillo*, el paciente se ha caído.

4.0.6. Prueba 3-1: Comportamiento despistado repetir explicación en el Twister

Archivo de entrada: **prueba-3.clp**

Archivo de salida: **salida-prueba-3-1.txt**

Esta prueba comprueba que ante un paciente con personalidad despistado, el sistema genera la repetición de la explicación del juego. Esto puede ocurrir 0, 1 o más veces seguidas. No hace falta realizar una prueba para el juego del **Tres en Raya** porque sigue el mismo esquema.

4.0.7. Prueba 3-2: Comportamiento despistado repetir movimiento en el Twister con finalización normalArchivo de entrada: **prueba-3.clp**Archivo de salida: **salida-prueba-3-2.txt**

En esta prueba se comprueba que el paciente despistado no realiza el movimiento que le dice el robot y por tanto este le repite el último movimiento. Esta prueba concluye con la finalización de la partida con normalidad, es decir, que el paciente sigue jugando y termina correctamente.

4.0.8. Prueba 3-3: Comportamiento despistado repetir movimiento en el Twister con finalización por demasiados despistesArchivo de entrada: **prueba-3.clp**Archivo de salida: **salida-prueba-3-3.txt**

Esta prueba es similar a la anterior, pero en este caso el paciente despistado no realiza el movimiento que le dice el robot más de 6 veces y por ello la partida se acaba antes.

4.0.9. Prueba 4-1: Comportamiento despistado repetir movimiento en el Tres en Raya con finalización normalArchivo de entrada: **prueba-4.clp**Archivo de salida: **salida-prueba-4-1.txt**

En esta prueba donde se juega al **Tres en Raya** el paciente con personalidad despistado se olvida de colocar ficha y el robot se lo recuerda. Esta prueba finaliza de forma normal.

4.0.10. Prueba 4-2: Comportamiento despistado repetir movimiento en el Tres en Raya con finalización por demasiados despistesArchivo de entrada: **prueba-4.clp**Archivo de salida: **salida-prueba-4-2.txt**

Es una prueba similar a la anterior, con la diferencia de que en este caso el paciente se despista más de 6 veces y por tanto la sesión termina antes.

Nota: dado que la probabilidad de despistarse es pequeña, y además va decreciendo a lo largo de la partida, porque se asume que cuanto más juegue menos despistado estará el paciente, ha sido necesario aumentar la probabilidad para poder realizar esta prueba.

4.0.11. Prueba 5-1: Comportamiento enérgico repetir el juegoArchivo de entrada: **prueba-5.clp**Archivo de salida: **salida-prueba-5-1.txt**

En esta prueba se simula que el paciente enérgico, después de una partida de **Tres en Raya** sigue teniendo energías, y quiere jugar otra vez. Esto puede ocurrir hasta un máximo de 4 partidas.

4.0.12. Prueba 5-2: Comportamiento enérgico impedir colocar fuera de turno en el Tres en Raya con finalización normalArchivo de entrada: **prueba-5.clp**Archivo de salida: **salida-prueba-5-2.txt**

Esta prueba sirve para determinar si se realiza correctamente las regañinas o reproches del robot hacia el paciente enérgico cuando este intenta colocar fuera de turno.

Nota: Esto ocurre en la partida 3 del archivo de salida

4.0.13. Prueba 5-2-bis: Comportamiento enérgico impedir colocar fuera de turno en el Tres en Raya con finalización por demasiadas trampasArchivo de entrada: **prueba-5.clp**Archivo de salida: **salida-prueba-5-2.txt**

En este caso, la salida es similar a la anterior, pero en este caso el paciente enérgico comete demasiadas trampas, y se le da la partida por perdida.

Nota: Esto ocurre en la partida 4 del archivo de salida

4.0.14. Prueba 6: Comportamiento enérgico repetir historial en el TwisterRaya

Archivo de entrada: **prueba-6.clp**

Archivo de salida: **salida-prueba-6.txt**

Esta última prueba consiste en comprobar cuando el paciente enérgico cambia de posición completamente. Cuando esto ocurre el robot le repite la secuencia entera de acciones hasta el momento para que el paciente se pueda recolocar.

5. Conclusiones

En esta práctica se ha tenido que realizar por primera vez una programación algo más compleja, al tenerse que resolver un problema mucho mayor a los resueltos previamente en los talleres. Aunque la metodología que se ha utilizado es la misma, las dimensiones de este problema son mucho mayores y esto ha hecho que se haya tenido que crear un mayor número de clases y reglas.

Aún así, ha sido muy útil llevarla a cabo, pues se han podido afianzar más los conocimientos de los sistemas de producción. Pudiendo ver su aplicación a problemas muy cercanos a la realidad.

A pesar de ello, sigue siendo un problema relativamente reducido, pero que podría ser escalable a un problema real, para el cuál simplemente haría falta ampliar el número de reglas, y aumentar el tamaño de la ontología en consecuencia.

Se piensa que para el problema planteado en concreto, la solución obtenida es bastante completa y sofisticada, aunque, tal y como se menciona en la práctica, la capacidad de generalización del código es reducida.

6. Comentarios personales

Respecto a problemas encontrados durante la realización de la práctica, simplemente destacar que en un primer momento se enfocó mal el desarrollo de la ejecución del programa. Se crearon reglas que guiaban la sesión de principio a fin, lo cual no tenía sentido, ya que este desarrollo se tenía que hacer con reglas que se fueran ejecutando de manera aleatoria según se avanzaba en la sesión.

Esto hizo que se tuviera que empezar de nuevo, pues se tuvo que modificar la ontología y crear todas las reglas. Al principio, resultó ligeramente confuso, pero una vez se comprendió la dinámica del sistema, y su funcionamiento, el resto fue relativamente sencillo.

En cuanto a comentarios personales, decir que ha sido una práctica especialmente interesante para adecuarnos a un paradigma de programación que hasta el momento nos era desconocido.

Como única crítica constructiva, mencionar que el taller de *la fábrica de cerveza*, al ser un problema que generaba acciones de forma muy guida, hizo que se creyera en un primer momento que se tenía que realizar de la misma manera. Esto no fue un buen planteamiento inicial, aunque al final se terminó enfocando la práctica correctamente.