uc3m | Universidad Carlos III de Madrid

PRÁCTICA I. Sistemas de Producción

Ingeniería Del Conocimiento

Curso 2021-22, Ingeniería Informática

Marly Irala Segovia

100405994@alumnos.uc3m.es Leganés. G84

Jaime Núñez Delgado

100383416@alumnos.uc3m.es Leganés. G84

Índice

Introducción **2**

Manual Técnico **2**

Ontología 2

Reglas 2

Manual de Usuario **3**

Pruebas realizadas **3**

Conclusiones **3**

Comentarios personales **3**

Introducción

En este documento hablaremos sobre nuestra implementación de un sistema de producción que es capaz de diseñar y simular un juego entre un robot y un niño.

Primero hablaremos sobre las reglas y la ontología, así como las decisiones de diseño que hemos tenido en cuenta para llegar a elegirlas. Seguido de esto describiremos y hablaremos sobre el funcionamiento del programa realizado.

Describiremos también las pruebas que hemos realizado al igual que el análisis de los resultados obtenidos por estas. Y por último mostraremos nuestras conclusiones y comentarios personales extraídos de la realización de la práctica.

Manual Técnico

Ontología

Concepto	Características	Valores
Juego	tipo	oca
		rayuela
	elemento	dado
		piedra
	max-casillas	Integer
	max-rondas	Integer
Elemento	valor	Integer
	nombre	niño
		robot
	personalidad	torpe
		tramposo

Jugador		impaciente
	posición	Integer
	turno	si
		no
	num-rondas	Integer
	num-haceMal	Integer
	num-haceMal-max	Integer
Casilla	tipo	oca
		puente
		muerte
	inicio	Integer
	fin	Integer
HaceMal	personalidad	torpe
		tramposo
		nervioso
	acción	String
Corrige	jugador	niño
	acción	String
	respuesta	String

Reglas

Comunes

Bienvenida
Esta regla imprime las características principales de la simulación si hay un juego y un niño con el que jugar. Para que se ejecute solo una vez al principio de cada simulación establecemos cierta prioridad y creamos un hecho una vez se ejecuta, de esta forma evitamos que se vuelva a ejecutar.

Acciones
Esta regla muestra por pantalla las acciones que puede realizar el niño en función de su personalidad. Hemos decidido separarla de la regla de bienvenida ya que al incluir más de una acción, esta regla se equipara con la combinación de estas y se ejecuta más de una vez. De la forma que lo hemos planteado va ejecutando todas las acciones posibles una a una y una sola vez.

TirarElemento
Con esta regla se puede tirar tanto el dado como la piedra. Simplemente se comprueba que no se ha tirado todavía ningún elemento y escoge un valor aleatorio para este. Una vez tenemos el valor del dado o piedra creamos un hecho de que el elemento está tirado.

Al principio del diseño lo que hacíamos era escoger un número aleatorio cuando se movía, pero haciendo esto se podría dar el caso de que el niño haga una cosa mal antes de que se tire ningún elemento. Sin embargo, creando esta nueva regla evitamos eso y simplificamos el código ya que realizamos lo mismo tanto para moverse en la oca como en la rayuela.

elNiñoHaceAlgoMal

En principio, habíamos definido la ontología de tal forma que el número de veces que una acción se hacía mal lo definimos en la clase HACEMAL,es decir, cada acción tenía un contador del número de veces que ocurría. Sin embargo, nos dimos cuenta de que era mejor hacerlo independientemente de la acción de forma que cambiamos el contador del número de veces que hacía algo mal a la clase JUGADOR.

Como se ha comentado anteriormente, para evitar que el niño haga algo mal primero comprobamos que se haya tirado un elemento.Si el niño hace algo mal se aumenta en uno el número de veces del jugador niño, se actualizan los turno ya que el niño pierde el turno y se imprime tanto la acción que ha realizado el niño como la corrección que realiza el robot.

Hemos decidido incluir la corrección en la misma regla de que el niño realiza algo mal ya que inicialmente lo hacíamos en dos reglas distintas y no parecía lo más eficiente. En esta segunda regla lo único que hacía era imprimir la corrección después de comprobar que el niño había hecho una mala acción, por esto decidimos que era mucho más sencillo hacerlo nada más realice la acción. Así también evitamos utilizar hechos y reducimos una regla.

seAcabaelJuegoLimiteHaceMal
Teniendo en cuenta que nos puede interesar acabar el juego antes dependiendo de la personalidad, hemos creado la regla de que si el niño hace algo mal un número determinado de veces durante una misma partida, se acaba el juego.

seAcabaElJuego
En cada sesión se jugará un número determinado de rondas del mismo juego, al llegar al máximo, se terminará la sesión. Se imprimirá que el juego se ha acabado y quién es el ganador.

Oca

moverOca
En esta regla el jugador al que le toque tirará el dado y sumará a su casilla actual el valor del dado avanzando su posición a la casilla resultante. Se imprimirá el valor del dado y la actualización de la casilla del jugador.

moverCasillaEspecial
Si el jugador cae en una de las casillas especiales (oca, puente o muerte), su posición cambiará automáticamente a una nueva casilla. Se imprimirá que el jugador ha caído en una casilla del tipo correspondiente y de donde a donde ha cambiado su posición.

acabarOca
Si la casilla de un jugador es igual o mayor al número máximo de casillas del juego, aumentamos en uno el número de rondas que hace el jugador. También se reinicia la posición de los dos jugadores para que de esta forma se pueda jugar más de una ronda.

Rayuela

moverRayuela
Se simula tanto el lanzamiento de la piedra a una casilla aleatoria como el movimiento del niño. Se imprime como este salta por encima de la piedra sin pisarla y llega hasta el final, también como a la vuelta se para para recoger la piedra y como vuelve con ella a la casilla de salida. También se actualiza el turno.
Hemos decidido hacer en una misma regla que el jugador completa una ronda (ida y vuelta) ya que al no avanzar de uno a uno, no tenía sentido tener otra regla solo para imprimir que está volviendo haciendo prácticamente las mismas comprobaciones que en la regla de ida, por esto hemos incluido ambas reglas en una sola.

Manual de Usuario

Sesión

Para empezar cada simulación primero será necesario cargar los ficheros correspondientes con la ontología, las reglas y las instancias de cada simulación en la herramienta CLIPS, ejecutar "reset" y finalmente "run".

Al empezar la sesión se tiene que mostrar por pantalla cuál ha sido el juego elegido y cuál es la personalidad del niño. Después tiene que aparecer las acciones asociadas a cada personalidad..

Oca

Para jugar al juego de la oca será necesario definir las instancias correspondientes con este juego, es decir, poniendo el tipo de juego como oca y estableciendo el elemento como dado. El resto de parámetros es acorde a lo que se quiere simular en dicho juego.

El funcionamiento del juego es el siguiente, los dos jugadores empezarán en la casilla de salida e irán tirando el dado una vez cada uno, el dado va desde uno hasta seis, el valor que salga en el dado se le sumará a la posición del jugador. Si cae en una de las casillas especiales, ocurre lo que se describe en cada una de estas. Si alguno de los jugadores consigue llegar al final se acabará la partida. Si el niño hace algo mal un determinado número de veces, la partida también terminará.

El puente: Al caer en esta casilla el jugador avanzará o retrocederá automáticamente a otra casilla.
La oca: Al caer en esta casilla, el jugador avanzará automáticamente a otra casilla. No volverá a tirar, como pasa en el juego real.
La muerte: Cuando el jugador aterrice en esta casilla, pasará directamente a la casilla de salida sin contar como una victoria y tendrá que volver a empezar la partida desde ahí.

Rayuela

Para jugar al juego de la oca será necesario definir las instancias correspondientes con este juego, es decir, poniendo el tipo de juego como rayuela y estableciendo el elemento como piedra. El resto de parámetros es acorde a lo que se quiere simular en dicho juego.

Para la simplificación de este juego, los jugadores tiran la piedra a una casilla aleatoria y avanzan hasta la anterior, saltan por encima de la casilla de la piedra, llegan hasta el final y vuelven recogiendo la piedra por el camino. Al terminar, será el turno del siguiente jugador. Para acabar la partida se tendrán que realizar tres rondas.

Pruebas realizadas

Nota: las pruebas se han realizado en windows debido a que en ubuntu no nos funcionaba correctamente el random.

1. **Juega a la oca**
   En esta primera prueba comprobamos que la simulación funciona correctamente con las instancias de un juego normal. En este caso juega un niño tramposo que puede moverse una casilla de más o tirar dos veces el dado, se gana al completar una ronda.

   En el fichero de salida se puede ver como los turnos y la actualización de la posición funcionan correctamente, en este caso el ganador es el robot.

2. **Juega a la rayuela**
   Probamos ahora el juego de la rayuela, nos aseguramos el correcto funcionamiento con las instancias de un juego normal. El niño es un niño torpe que puede caer o pisar la raya al saltar, y para ganar una partida se tienen que completar 3 vueltas.

   En el fichero de salida podemos ver que la simulación se realiza correctamente, respetando los turnos y mostrando al ganador una vez se completan las 3 rondas. Nos damos cuenta de que el niño realiza una acción mal las tres veces que le toca el turno y no consigue completar ninguna vuelta, para evitar esto realizaremos otra prueba donde aumentemos el número de rondas.

3. **Juega a la rayuela con 15 rondas**
   Para esta tercera prueba simulamos un niño muy torpe que juega una partida de rayuela más larga, con 15 rondas.
   Como se ve en el fichero de salida, esta vez el niño ya puede completar un par de vueltas aunque se sigue equivocando varias veces. Para evitar que se equivoque tantas veces realizaremos otra prueba donde limitamos el número de veces que hace mal algo.

4. **Niño impaciente que quiere que se acabe ya la partida de rayuela**
   Para simular esta prueba definimos un niño impaciente que pide de forma aleatoria que quiere que se acabe ya la partida. Para controlar esto establecemos que el número máximo que pueda pedir que se acabe la partida sea dos y que cuando llegue al límite se acabe el juego para que no se ponga nervioso. Para esta simulación el juego acaba en 5 rondas.
   Como se puede observar en el fichero de salida, el robot para el juego después de que el niño lo pida dos veces.

5. **Niño impaciente que no quiere jugar el juego de la oca completo**
   Para simular esta prueba definimos un niño impaciente que quiere jugar a la oca, pero como se le hace largo la oca con 62 casillas reducimos el nº de casillas a la mitad (31 casillas).
   En el fichero de salida vemos como el juego acaba cuando la posición del robot llega a la casilla 31.

6. **Niño le apetece jugar otra ronda de la oca**
   En esta cuarta prueba se simula un niño tramposo que quiere jugar dos rondas de la oca en vez de una.
   Vemos como la partida se acaba cuando uno de los jugadores completa las dos vueltas, por tanto, esta simulación también funciona correctamente.

Conclusiones

Es difícil tener una visión general y completa de cómo quieres realizar la práctica antes de empezarla ya que a medida que la íbamos haciendo nos íbamos dando cuenta de los errores que cometíamos. A medida que se avanzaba con la realización de la práctica era preciso cambiar cosas que antes se creían que estaban bien, tanto reglas como la propia ontología.
Intentando reducir las reglas hemos tenido que añadir más atributos a las clases pero computacionalmente es una mejor solución.
Por último mencionar, que la complicación principal de esta práctica consistía en generalizar las reglas para ambos juegos y no tanto el hecho de simular cada juego por separado.

Comentarios personales

Es una práctica muy completa en cuanto al contenido que hemos visto durante el curso hasta ahora y útil para acabar de comprender el funcionamiento de la herramienta CLIPS de una forma amena. Consideramos también que los talleres han sido muy útiles para llevar la práctica al día, lo cual creemos que es algo muy acertado.