Práctica de Sistemas de Producción

DOBLE GRADO INFORMÁTICA Y ADE
JUAN ORTEGA SÁLINAS 100332835
DAVID PALOMERO MÉNDEZ 100333208

Introducción

En este documento vamos a presentar la resolución de la práctica de la asignatura de Ingeniería del Conocimiento. La práctica consiste en hacer un sistema de producción basado en reglas sobre el juego de mesa Agrícola. Este sistema de producción será realizado en el lenguaje de programación CLIPS y será capaz de jugar partidas a este juego en modo individual. Es importante mencionar, que nosotros realizaremos una aproximación del problema, la cual no representará la totalidad de este.
La estructura del documento será la siguiente:

- Primero realizaremos una descripción del código implementado, explicando las reglas y la ontología. Este apartado será el manual técnico
- En segundo lugar, realizaremos una descripción del uso del programa que hemos realizado, es decir, un manual de usuario para que cualquier persona que no haya desarrollado el programa pueda usarlo.
- En tercer lugar, realizaremos una serie de pruebas para verificar el correcto funcionamiento del programa y un análisis de los resultados obtenidos.
- Por último, unas breves conclusiones acerca de la realización de la práctica.

Hemos desarrollado una estrategia por turnos para optimizar al máximo las posibilidades, con la intención de conseguir el objetivo de la forma más fácil y eficiente posible. Esto nos permite conseguir más puntos en el juego dependiendo del orden en el que aparezcan las acciones.

Manual Técnico

En este apartado explicaremos la ontología que hemos creado y usado, así como las reglas. En este apartado realizaremos una descripción detallada del código implementado. Este código implementado consta de dos partes: la ontología, almacenada en el archivo "ontologia.clp" y las reglas del archivo "reglas.clp". En el archivo de la ontología se encuentran las clases necesarias para el funcionamiento, así como las plantillas auxiliares. En el archivo de las reglas se encuentran todas las reglas para poder jugar una partida del juego Agrícola teniendo en cuenta que nuestro objetivo es fallar lo menos posible al dar de comer a los habitantes.
Utilizamos los hechos como *Vacios*, *Habitantes*, y *Contador* para ayudarnos a la hora de realizar el programa. Estos hechos están definidos en todos los archivos de prueba en el deffacts *HechosIniciales*.

La ontología está formada por seis clases y dos plantillas auxiliares. Las clases son las siguientes: *EspacioAnimales*, *EspacioCampo*, *EspacioHabticacion*, *AdquisicionMayor* y Acciones. Las plantillas son *Almacenado*, *InfoJuego* y *AccionesDisponibles*. Comenzaremos a explicar una a una cada clase. Una vez acabemos con las clases explicaremos brevemente las plantillas.
La clase *EspacioAnimales* nos sirve para representar la información relacionada con los espacios destinados para guardar animales. Los atributos de esta clase serán:

- **Tamaño**: para controlar el número de espacios por los que está formado, en este caso uno o dos.
- **Establos**: para controlar el número de establos que hay en el espacio. Al igual que en el tamaño puede haber ninguno, uno o dos, dependiendo del tamaño del espacio.
- **Vallas**: para saber cuantas vallas tiene el espacio y poder controlar si es un espacio cerrado o no
- **Habilitado**: para saber si esta cerrado o no y, por lo tanto, saber si podemos colocar animales o no
- **Capacidad**: para saber el número máximo de animales de un mismo tipo que tenemos que podemos tener
- **Ocupación**: para saber el número actual de animales en el espacio
- **Animal**: para saber el tipo de los animales almacenados en este espacio

La clase *EspacioHabitacion* lo utilizaremos para representar los espacios ocupados por habitaciones en el tablero, solo nos interesa saber el tipo del espacio y si contiene o no un miembro de la familia.

- **Tipo**: Hace referencia al tipo de la habitación en concreto que puede ser de piedra, madera o adobe.
- **Habitante**: Tendrá el valor True cuando la habitación tenga un miembro de la familia y False en caso contrario.

La clase *EspacioCampo* nos sirve para saber si un espacio del tablero del jugador es de tipo campo y por tanto se puede sembrar.

- **Tipo**: Es el tipo de objeto que está sembrado en el espacio concreto, puede ser cereal, hortalizas o ninguno en caso de que esté vacío.
- **Cantidad**: nos indica la cantidad de objetos que hay en el campo sembrado.

La clase *Adquisiciones mayores* contiene la información asociada a cada tipo de adquisición mayor.

- **Puntos**: Son los puntos que otorga al final de la partida por haber sido adquirida.
- **Adobe**: número de adobes requiere la compra.
- **Madera**: número de maderas requiere la compra.
- **Juncal**: número juncales requiere la compra.
- **Piedra**: número de piedras requiere la compra.
- **Tipo**: hogar, cocina, pozo, cestería, alfarería, horno de adobe, ebanistería y horno de piedra: Tipo de adquisición, para asociar cada tipo de acción.
- **Disponible**: Con esto controlaremos si queda alguna disponible o no.

En la clase *Acción* representaremos la información de las acciones.

- **Nombre**: Nombre de la acción.
- **Disponible**: True si se ha desbloqueado la acción, False en caso contrario.
- **Utilizado**: True si ya ha sido utilizada en un turno, False en caso contrario.
- **Cantidad**: Es la cantidad de recursos que contiene la acción.
- **Recolocar**: Es la cantidad de recursos que se tienen que acumular en cada ronda.
- **Recolocado**: True si ya se han aumentado los recursos necesarios, False en caso contrario.

En la clase *Almacenado* representamos la información correspondiente sobre nuestro almacén.

- **Tipo**: Tipo de material del que se tiene información.
- **Cantidad**: Cantidad del material almacenada.

La plantilla *InfoJuego* representa la información de el turno y la fase.

- **Turno**: Referencia el número de ronda
- **Fase**: Referencia el número de fase.

La plantilla *AccionDisponible* la usamos para indicar cuándo podrán estar disponibles las acciones.

- **Nombre**: Nombre de la acción
- **Turno**: Turno en el que la acción se volverá disponible.

La plantilla *Habitantes* la usaremos para representar el número de habitantes y cuántos de ellos son recién nacidos.

- **Total**: Número total de habitantes.
- **Nacidos**: Número de recién nacidos.

La plantilla *AnimalColocar* la usaremos para saber cuándo hay que colocar animales.

- **Tipo:** Tipo de animal que vamos a colocar.
- **Cantidad:** Número de animales que vamos a colocar.

La plantilla *AnimalVender* la usaremos para saber cuándo hay que colocar animales.

- **Tipo**: Tipo de animal que vamos a vender.

- **Cantidad**: Número de animales que vamos a vender.

En segundo lugar, explicaremos todas las reglas, incluidas las auxiliares, que aparecen en el fichero “Reglas.clp”. Queremos mencionar, antes de empezar la explicación, que no van a aparecer en el fichero ciertas reglas correspondientes a diversas acciones porque no las usaremos para cumplir nuestro objetivo.

Comenzaremos por las reglas que consideramos principales, es decir, las reglas que realizan las acciones. Estas reglas son las siguientes:

- **ObtenerMadera**: sirve para realizar la acción “Bosque”. Esta acción consiste en coger toda la madera que esté almacenada en dicha acción. Para ello, modificamos la instancia de acción correspondiente para dejar a 0 la cantidad y, al mismo tiempo, modificamos la clase almacenado para guardar la cantidad obtenida. Además, en la clase acción también modificamos el atributo utilizado para controlar que esa acción ya se ha realizado.
- **ObtenerAdobe**: esta regla es exactamente igual a la de *ObtenerMadera* pero con la materia prima adobe.
- **ObtenerJuncal**: esta regla es exactamente igual a la de *ObtenerMadera* pero con la materia prima adobe.
- **ObtenerPiedra**: esta regla es exactamente igual a la de *ObtenerMadera* pero con la materia prima piedra.
- **SemillasCereales** y **SemillasHortalizas**: estas reglas tienen la misma estructura que *ObtenerMadera* con la diferencia de la materia prima. En un caso, con la acción se obtiene una semilla de cereal y en el otro se obtiene una de hortaliza.
- **Pesca** y **Jornalero**: estas reglas son iguales a las demás reglas de obtener materia prima. En este caso, con ambas acciones obtienes comida.
- **AmpliarHabitación**: esta regla sirve para ampliar en una el número de habitaciones. Para ello, realizamos las mismas comprobaciones que en las demás reglas (si esta disponible y si ya se ha utilizado). Además, comprobamos si el jugador tiene los materiales necesarios y si hay espacios vacíos para colocar la habitación. De cumplir todas las condiciones, se crea una instancia de *EspacioHabitacion* y se reduce en uno el contador de espacios vacíos borrando y añadiendo el hecho Vacíos.
- **Cocina**: esta regla sirve para la obtencion de la adquisicion mayor cocina. Las condiciones que tiene que cumplir son que esté disponible la acción y que no la estén utilizando y que el jugador tenga el coste necesario. Si las cumple se modifica el atributo adquirido de la instancia de *AdquisicionMayor* y se marca la acción como utilizada.
- **AmpliacionPlanificada**: con esta acción se puede aumentar el número de habitantes en uno si disponemos de una habitación libre, es decir, que no haya ningún habitante en ese *EspacioHabitacion*.
- **AmpliacionPrecipitada**: esta acción tiene el mismo efecto que la regla anterior. El único cambio que se produce es que no hace falta tener ninguna habitación disponible para ese nuevo habitante.
- **Vallas**: esta regla sirve para realizar la acción de colocar vallas. Esta acción hemos decidido simplificarla con respecto a la acción real para que sea más fácil de implementar. Lo que hemos hecho para simplificar la regla ha sido que cuando la

vayamos a realizar siempre colocamos 4 vallas y cerramos un espacio de animales de tamaño 1.

- **MercadoOvino**: esta regla nos permite realizar la acción del juego Mercado Ovino. Esta regla lo que hace es crear dos hechos, uno para colocar dos animales si se puede y otro para vender los demás. En el caso de que no se puedan colocar esos dos animales también los venderemos. Lo hemos decidido así en función del objetivo establecido.
- **MercadoPorcino**: es exáctamente igual que *MercadoOvino* pero pero con cerdos.
- **MercadoBovino**: de nuevo igual que *MercadoOvino* pero con vacas.
- **Siembra**: esta regla nos permite sembrar un campo labrado. Hemos decidido sembrar solo Cereal porque las hortalizas nos producen comida con mayor facilidad y nos interesa tener más cereal para hornear pan.
- **Hornear**: esta regla permite ejecutar la acción hornear pan asociada a la cocina. Con esta acción podremos convertir 1 trigo en 2 de comida.

Tenemos unas reglas auxiliares que nos permiten realizar operaciones que son necesarias en el juego, pero que no tienen una acción asociada que requiera la utilización de un habitante.

- **ColocarAnimales**: nos permite colocar los animales obtenidos en los espacios delimitados por vallas que estén vacíos.
- **VenderOvejas**: esta regla nos permite utilizar la cocina para “vender” ovejas por comida. Cada oveja se convertirá en 2 de comida.
- **VenderCerdos**: hace lo mismo que la regla anterior, pero cada cerdo se convierte en 3 de comida.
- **VenderVacas**: igual que las anteriores, cada vaca se vende por 4 de comida.

También tenemos las reglas para poder ejecutar el proceso de cosecha que se da en la fase 4 del juego:

- **Cosecha**: esta es la regla principal que gestiona todo. Comprueba si se está en el turno y la fase correctas y calcula la comida que necesitamos para dar de comer a los habitantes. Comprueba si tenemos comida suficiente para dar de comer a todos los habitantes y genera los hechos necesarios para proceder con los siguientes pasos.
- **VenderHortalizasParaComer**: esta regla nos permite cambiar 1 hortaliza por 3 de comida mediante la cocina y nos servirá para obtener más comida en caso de no poder dar de comer a los habitantes.
- **VenderOvejasParaComer**: igual que la regla anterior pero cada oveja produce 2 de comida.
- **VenderCerdosParaComer**: es como las dos reglas anteriores, pero utilizamos los cerdos para cambiar cada uno por 2 de comida
- **VenderVacasParaComer**: como las acciones anteriores pero cambiando cada vaca por 3 de comida.
- **RecogerOvejaCosecha**: produce una oveja siempre y cuando tengamos una pareja de ovejas previa, la oveja que recoge la cambiamos por comida.
- **RecogerCerdoCosecha**: igual que la regla anterior pero con cerdos.
- **RecogerVacaCosecha**: igual que las anteriores, utilizando vacas.
- **RecogerCereal**: regla para comprobar los campos con cereal sembrado y recoger un cereal de cada campo sembrado.

- **SinComida**: si no hemos conseguido dar de comer, guardamos cuantos fallos hemos cometido para restar los puntos.

En segundo lugar, tenemos las reglas que nos ayudan a decidir qué hacer en cada turno. Estas reglas tienen en el nombre el número del turno al que pertenecen. Antes de implementar estas reglas hemos establecido una estrategia para llegar a cumplir nuestro objetivo. Con estas reglas, implementamos la estrategia de juego que vamos a seguir. Cada turno se decidirá qué acción hay que realizar en función de las acciones disponibles en ese momento, la situación de los espacios y las materias primas en ese momento, especialmente la comida.
Hemos implementado tres reglas auxiliares por si es necesario completar acciones en alguno de los turnos. Estas reglas son *JornaleroExtra*, *SemillaHortalizaExtra*y *SemillaCerealExtra*, que realizan las acciones *Jornalero* y *SemillaCereales* respectivamente.

Manual de usuario

En este apartado realizaremos una descripción del uso del programa que hemos realizado. Para poder utilizar nuestro programa deberá tener instalado el interpretador del lenguaje de programación CLIPS. Cuando se tenga instalado el interpretador, el usuario deberá abrir los archivos “Ontologia.clp” y “Reglas.clp”, en este orden. Una vez se hayan abierto los dos archivos, se deberá crear un tercer archivo de pruebas para usar el programa con la misma extensión, .clp. Este tercer archivo tendrá que ser creado y modificado por los usuarios. El archivo de pruebas contendrá las instancias de la clase *AdquisicionMayor* que hemos usado en nuestra solución (la cocina más barata de las dos), las instancias de la clase *Accion*, los hechos de la plantilla *AccionDisponible* y el hecho de la plantilla InfoJuego que empezará en el turno 1 y fase 1. Los hechos de la plantilla *AccionDisponible* podrán ser alterados de orden dentro del periodo al que pertenecen para tener diferentes trazas de ejecución. El nombre de las instancias de *Accion* y los hechos de *AccionDisponible* tendrán que tener el mismo nombre si la acción es la misma.

Estas son todas las acciones que se irán habilitando durante los turnos del juego.

Una vez estén todos los ficheros cargados, el usuario tendrá que poner los comandos (reset) y (run) para que se ejecute el programa. El programa creará un fichero de texto

llamado “salida-PruebaX.txt”, donde aparecerá la traza de ejecución del programa, es decir, todas las impresiones de todas las reglas que se ejecuten estarán escritas en el fichero.
Queremos recordar que nuestra solución es una aproximación del problema y que nuestro objetivo no es conseguir el máximo número de puntos posibles, sino que fallemos lo menos posible al dar de comer a los habitantes.

Pruebas realizadas

En este apartado realizaremos una descripción del plan de pruebas que hemos creado para verificar el correcto funcionamiento de nuestro programa. Posteriormente realizaremos un análisis de los resultados obtenidos. De igual manera que ocurre con las reglas, no crearemos todas las instancias de las adquisiciones mayores porque no las usaremos, así como algunas acciones.

- **Prueba 1:**

En este caso todas las acciones de animales salen en los primeros turnos de los periodos en los que serán habilitadas y las acciones de ampliar familia igual. De esta forma lo que se consigue es acumular el mayor número de animales posible y poder hacer el mayor número de acciones durante la partida.
El objetivo se cumple en esta prueba, es decir, damos de comer a todos los habitantes teniendo en cuenta el orden de aparición de las acciones del tablero.

- **Prueba 2:**

Esta prueba consiste en que las acciones de animales y las acciones de ampliación familiar se habiliten en el último turno del periodo correspondiente, acumulando muy pocos animales y perdiendo acciones realizables durante la partida.
Según este orden de acciones, no hay problema para dar de comer a todos los habitantes, por tanto, cumplimos el objetivo.

- **Prueba 3:**

Es un caso en el cual se acumularán muchos animales, pero se dejarán de hacer algunas acciones durante la partida, debido a la imposibilidad de ampliar la familia antes.
La prueba 3 también es solventada satisfactoriamente por el programa, siendo este capaz de dar de comer a todos los habitantes.

- **Prueba 4:**

En esta prueba hemos querido recrear una situación en la que se podrá ampliar la familia lo antes posible en cada periodo correspondiente, pero el número de animales que se acumulan durante la partida será bajo.
En este caso, no cumplimos el objetivo planteado, ya que durante la ejecución no llegamos a dar de comer a todos los habitantes.

- **Prueba 5:**

En la última prueba buscamos la aleatoriedad máxima de las acciones, para comprobar otro comportamiento distinto.

Al ejecutar la prueba comprobamos que no falla al dar de comer, por tanto, se cumple el objetivo.

De 5 pruebas que hemos realizado, el programa solo ha fallado en su objetivo en 1. El fallo cometido en la prueba cuatro se debe a que las acciones de ampliar familia se desbloquean en los primeros turnos del periodo en el que se pueden obtener y a su vez, los mercados de animales se desbloquean en el último turno del periodo correspondiente a su desbloqueo, por tanto, el número de animales que se acumula es muy inferior al de otras pruebas y esto provoca que podamos obtener mucha menos comida mediante la cocina.

Conclusiones

Para llegar a una solución del sub-problema de “Agrícola 1 jugador” hemos utilizado el lenguaje CLIPS, como se nos pide en el enunciado. Al empezar a desarrollar la ontología nos dimos cuenta de la complejidad del problema al que nos estábamos enfrentando y por eso decidimos elegir un sub-problema que fuese bastante representativo. El sub-problema elegido consiste en fallar en dar de comer a los habitantes las menos veces posible.
Para incrementar un poco la dificultad hemos decidido dar máxima prioridad a las ampliaciones de familia, de modo que haya más habitantes a los que dar de comer y por tanto tengamos que optimizar más los recursos.
Se han implementado prácticamente todas las acciones del juego porque hemos intentado que se pudiese ampliar el problema si fuera necesario. Hay ciertas acciones que hemos simplificado para hacer el problema más asequible, ya que nuestro objetivo no tiene que ver con los puntos y podemos centrarnos más en otras cuestiones.
Por ejemplo, la colocación de animales y el cambio de animales por comida (mediante la cocina) se realizará de dos en dos, para simplificar las reglas y que el funcionamiento sea más sencillo.
Hemos omitido todas las adquisiciones mayores salvo la cocina (de coste 4), ya que, al ser el juego de 1 solo jugador, siempre elegiremos lo que más nos convenga, sin tener que hacer estrategias cambiantes en función de las interacciones con otros jugadores. Por esto hemos decidido implementar solo la cocina, que nos servirá tanto para “hornear pan” (que se puede realizar al utilizar la acción “Siembra”), como para cambiar el exceso de animales y hortalizas por comida. Esta última funcionalidad enlaza muy bien con el criterio de intercambiar los animales sobrantes por comida.