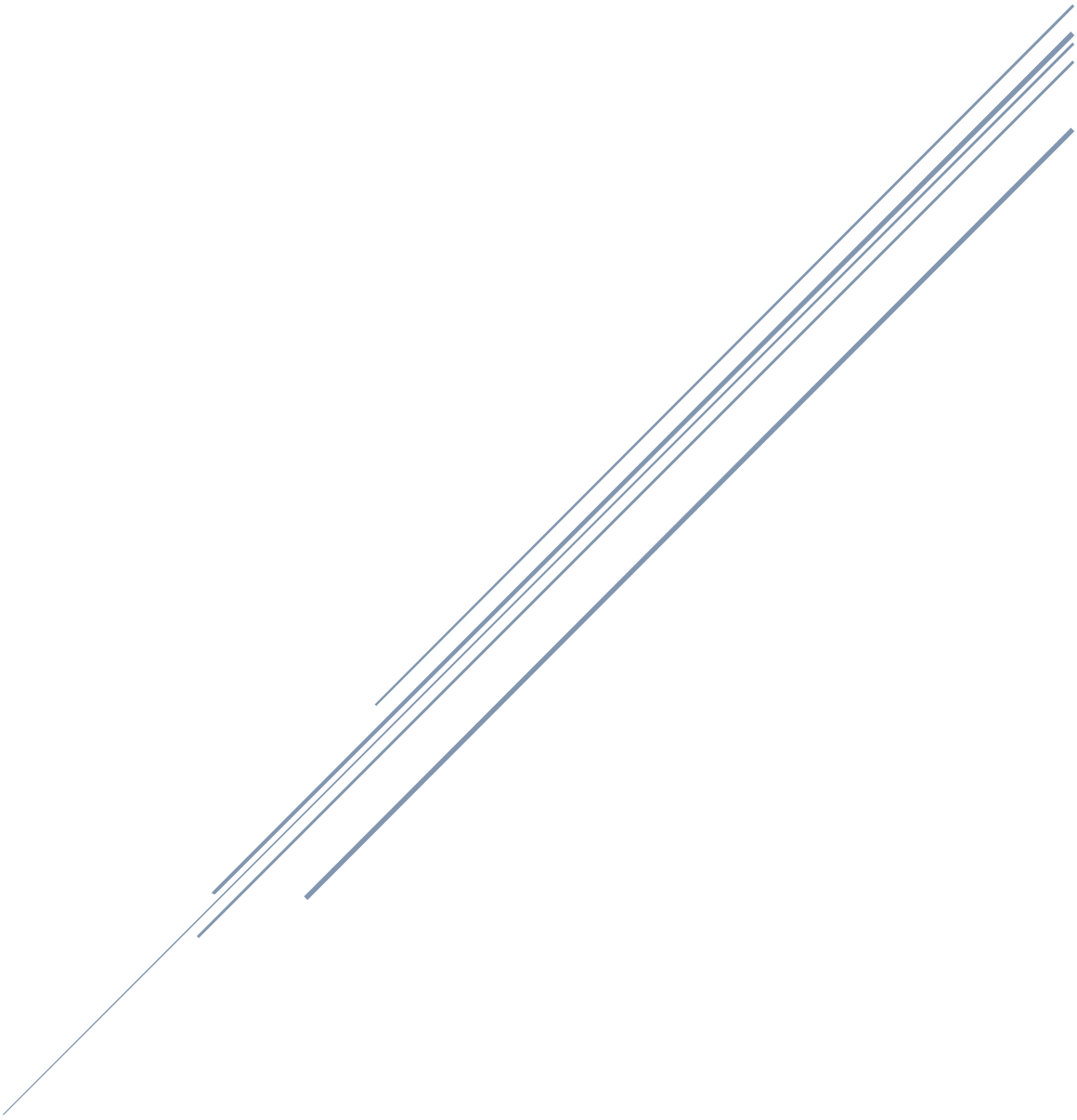
INGENIERIA DEL CONOCIMIENTO

PRÁCTICA 1, 4º curso

Universidad Carlos III Madrid, Ingeniería Informática, 2021



grupo 80 - Colmenarejo - IVAN AGUADO PERULERO – 100405871
grupo 80 - Colmenarejo - JORGE SERRANO PÉREZ – 100405987

CONTENIDO

1.- INTRODUCCIÓN

El objetivo de esta práctica es implementar un sistema de producción (SP) en CLIPS capaz de jugar partidas de forma inteligente a LeHavre. Este juego consiste en ser el jugador que más dinero acumule, ya sea en metálico o en propiedades, mediante la colocación de trabajadores y la construcción de distintos edificios.

En cuanto al presente documento, en primer lugar encontramos una introducción. A continuación, un manual técnico en el que se explicará el código implementado, es decir, las distintas reglas y la ontología. Lo siguiente será un manual de usuario que detalla una descripción del uso del programa realizado. Además, habrá otra sección para describir las pruebas que se han realizado para verificar su correcto funcionamiento y un análisis de los resultados obtenidos. Por último, se justificarán las conclusiones técnicas extraídas durante la realización de la práctica, así como distintos comentarios personales o problemas encontrados.

2.- MANUAL TÉCNICO

En este apartado se realizará una descripción detallada del código implementado, tanto las reglas específicas como la ontología. La solución trata sobre una aproximación al juego Le Havre con distintas relajaciones para facilitar su programación.

2.1.- ONTOLOGÍA

En este apartado se detallarán todas las clases definidas en el fichero *"ontología.clp"*, junto con las justificaciones de cada decisión tomada al implementar el programa. También se explicarán los atributos seleccionados para cada uno de los tipos de clases.

JUGADOR: representa un posible jugador participante del juego. En nuestra solución existen dos diferentes.

id: identificador del jugador.

casilla_ficha_barco: casilla en la que se encuentra el jugador dentro de la ronda. Hay 7 en total

disco_persona: edificio en el que se encuentra el jugador.

prestamos: número de préstamos que tiene el jugador.

francos: número de francos (dinero) que tiene el jugador.

riqueza_total: riqueza total del jugador, utilizada para dar un ganador. Incluirá los francos, el valor de los edificios y de manera negativa los préstamos.

ha_cosechado: indica si el jugador ya ha cosechado en caso de que al final de la ronda haya cosecha.

ha_pagado_comida: indica si el jugador ya ha pagado la comida al final de ronda.

BIENES: representa los posibles bienes del juego. Se han creado las instancias de todos los bienes para cada jugador (con su respectivo id) a pesar de que no tengan ninguna unidad del mismo.

id: identificador del jugador propietario del bien.

unidades: número de unidades del bien.

tipo: tipo del bien (puede ser Carbón, Arcilla, Madera, Piel, Acero o Vaca)

OFERTA: representa la oferta en bienes a añadir a las ofertas iniciales en una determinada casilla del tablero. En cada una de las 7 que hay tienen dos tipos de bienes distintos.

casilla: número de la casilla en la que se encuentra la oferta.

interes: indica si la oferta tiene intereses o no.

unidades_bienes: número de unidades del bien ofertado.

tipo_bienes: tipo del bien ofertado.

OFERTA_INICIAL: representa una determinada oferta de un tipo de bien, almacenada para ser recogida por uno de los jugadores. Hay una para cada tipo de bien, y solo la del Carbón tiene unidades al comenzar el juego.

unidades: unidades del bien ofertado.

tipo: tipo del bien ofertado.

EDIFICIO: representa los posibles edificios construidos o a construir del juego. Hemos creado 7 diferentes, 3 ya construidos por el Municipio y otros 4 que tendrán que construir los jugadores. Los municipales son los que tienen como acción la de construir el resto de edificios. Los pendientes por construir tienen todos como acción el obtener bienes de distintos tipos.

nombre: nombre identificador del edificio.

coste_construccion_unidades: número de unidades del bien necesario para la construcción del edificio. Puede darse el caso de que no tenga coste (0).

coste_construccion_tipo: tipo del bien necesario para la construcción del edificio. Puede darse el caso de que no tenga coste (Ninguno).

id: identificador del jugador propietario del edificio.

coste_entrada: número de francos a pagar por entrar al edificio.

valor_venta: precio en número de francos del edificio, tanto de compra como de venta.

construido: indica si el edificio está construido o no.

accion_unidades: número de unidades del bien asociado a la acción del edificio.

accion_tipo: tipo del bien asociado a la acción del edificio.

RONDA: representa las particularidades y el momento actual de cada ronda dentro del juego. Se han creado las 14 rondas que se indican en el juego en base al juego de 2 jugadores. La comida a pagar se va incrementando a lo largo de la partida y la cosecha puede o no producirse en cada ronda.

id: número identificador de la ronda (hay 14 rondas en total para el juego de 2 jugadores).

id_jugador_turno: identificador del jugador en turno.

cosecha: indica si hay cosecha en la ronda.

comida_a_pagar: indica la cantidad de comida a pagar al finalizar la ronda.

casilla: indica la casilla actual en la que se está jugando en la ronda.

jugando_ronda: indica si se está jugando o no la ronda.

2.2.- HECHOS

En este apartado se detallan una serie de hechos necesarios para la correcta ejecución de las reglas. Estos son *fase1_completada*, *fase2_completada*, *ha_comprado*, *hay_que_construir* y *hay_que_pagar_intereses*. Todos se usan como valores booleanos. Los dos primeros indican que la fase correspondiente del juego se ha completado: la primera es la de la oferta y la segunda la de la acción principal, que puede ser tomar los bienes de una oferta o mover a la persona para usar un edificio. El siguiente hecho indica si un jugador ha comprado un edificio y, por lo tanto, no puede venderlo en el mismo turno. El penúltimo sirve para realizar correctamente la acción de construir un edificio al tener a la persona en uno de los municipales (que son los que te permiten ejecutar la acción de construir). Y el último indica que hay que pagar los intereses en esa oferta para todos los jugadores.

2.3.- REGLAS

En este apartado se detallan cada una de las reglas implementadas, sus funcionalidades y la justificación de su diseño. Estas están divididas en diferentes subgrupos dependiendo de su aportación al correcto funcionamiento del juego.

En primer lugar, una condición necesaria para todas las reglas es que el jugador que va a realizar las acciones sea el correcto, es decir, aquel cuyo turno está indicado en la instancia de ronda. Por ello, está comprobación es general y se aplica en todas las reglas que mencionaremos a continuación.

INICIO EJECUCIÓN

inicio: asigna la estrategia de selección de reglas e indica el fichero donde se guardarán las trazas de la ejecución.

OFERTA

Aquí englobamos todas las reglas relacionadas con la fase 1, que se corresponde con añadir las ofertas a la oferta inicial correspondiente, es decir, a la cajita de cada bien en el juego físico. Por ello, todas ellas tendrán una comprobación de que la fase 1 no está completada, y al ejecutarse cambiarán el hecho indicando que ya está completada.

AñadirOfertaNoInteres: añade a las ofertas los bienes asociados a la casilla en caso de no haber intereses. Necesita 2 instancias de la oferta porque en cada casilla se añaden dos tipos de bienes diferentes.

AñadirOfertaInteresNoTienePrestamosJugador1: añade a las ofertas los bienes asociados a la casilla, en caso de haber intereses pero que el jugador en turno no tenga préstamos.

PagarInteresNoTienePrestamosJugador2: informa de que el jugador que no está en turno no ha tenido que pagar intereses al no tener préstamos.

AñadirOfertaInteresTienePrestamosJugador1: añade a las ofertas los bienes asociados a la casilla, en caso de haber intereses y que el jugador en turno tenga préstamos. A su vez el jugador mencionado paga los intereses.

PagarInteresTienePrestamosJugador2: el jugador que no está en turno paga intereses al tener préstamos.

AñadirOfertaInteresTienePrestamosNoDineroJugador1: añade a las ofertas los bienes asociados a la casilla, en caso de haber intereses, que el jugador en turno tenga préstamos y no dinero para pagarlos. A su vez el jugador mencionado paga los intereses pidiendo un nuevo préstamo.

PagarInteresTienePrestamosNoDineroJugador2: el jugador que no está en turno paga intereses al tener préstamos, teniendo que pedir uno nuevo.

TOMAR BIENES OFERTA

Aquí englobamos todas las reglas relacionadas con una posible acción de la fase 2, que es la de tomar los bienes de una oferta. Por ello, tendrán una comprobación de que la fase 1 ya está completada y la 2 no, y al ejecutarse cambiarán el hecho indicando que la fase 2 ya se ha completado.

TomarBienes: el jugador coge los bienes de una oferta.

TomarFrancosOferta: el jugador coge los francos de la oferta de francos. Es necesario dividir la regla en dos porque en nuestra implementación el jugador tiene un atributo específico que se corresponde con el número de francos, no se trata de un bien aparte.

MOVER JUGADOR A EDIFICIO

Aquí englobamos todas las reglas relacionadas con la otra posible acción de la fase 2, que es la de mover la persona a un edificio para ejecutar su acción. Por ello, tendrán una comprobación de que la fase 1 ya está completada y la 2 no, y al ejecutarse cambiarán el hecho indicando que la fase 2 ya se ha completado, de la misma manera que en las reglas anteriores.

MoverPersonaEdificioMunicipalInicialConstruido: el jugador se desplaza a un edificio del municipio de los iniciales (B1, B2 o B3) ya construido y recibe los beneficios de su acción. Estos son el poder construir un nuevo edificio (modifica el hecho *hay_que_construir*) y mantenerse como propietario.

MoverPersonaEdificioMunicipalNoInicialConstruido: el jugador se desplaza a un edificio del municipio no inicial ya construido y recibe los beneficios de su acción.

MoverPersonaEdificioJugadorConstruidoConAccion: el jugador se desplaza a un edificio de otro jugador ya construido y le paga para recibir los beneficios de su acción, mientras que la acción sea del tipo obtener bienes.

MoverPersonaEdificioJugadorConstruidoSinAccion: el jugador se desplaza a un edificio de otro jugador ya construido y le paga para recibir los beneficios de su acción, mientras que la acción no sea del tipo obtener bienes.

CONSTRUIR EDIFICIO

En este caso utilizamos el hecho *hay_que_construir* que se modificaba previamente para obligar a la ejecución de estas reglas y que se realice la acción correspondiente.

ConstruirEdificioConCosteTieneMateriales: el jugador construye uno de los edificios disponibles con coste de construcción, pues tiene los materiales necesarios.

ConstruirEdificioConCosteNoTieneMateriales: el jugador intenta construir uno de los edificios disponibles con coste de construcción, pero no tiene los materiales necesarios.

ConstruirEdificioSinCoste: el jugador construye uno de los edificios disponibles sin coste de producción.

COMPRAR EDIFICIO

Aquí englobamos todas las reglas relacionadas con la acción opcional de la fase 2 que es comprar un edificio. En este caso tendrán una comprobación de que la fase 1 ya está completada y la 2 también.

ComprarEdificioMunicipalVacio: el jugador compra un edificio del municipio construido que en ese momento se encuentra vacío.

ComprarEdificioMunicipalLleno: el jugador compra un edificio del municipio construido que en ese momento se encuentra lleno. De esta manera, el jugador que tenía la persona situada en este edificio deberá volver con su dueño, es decir, no asignarse a ningún otro edificio.

VENDER EDIFICIO

Aquí englobamos todas las reglas relacionadas con la acción opcional de la fase 2 que es vender un edificio. Por ello se realizan las mismas comprobaciones.

VenderEdificioJugadorVacio: el jugador vende un edificio de su propiedad construido al municipio que se encuentra vacío en ese momento.

VenderEdificioJugadorLleno: el jugador vende un edificio de su propiedad construido al municipio que se encuentra lleno en ese momento. De la misma manera que antes, la ficha de persona "vuelve" a su propietario.

PRÉSTAMOS

PagarPrestamos: el jugador paga un préstamo de los adquiridos. Se puede realizar en cualquier momento, por lo que no necesita muchas comprobaciones.

PedirPrestamo: el jugador pide un préstamo para no quedarse con un número de francos negativo al realizar algún tipo de acción.

TURNOS

CambiarTurno: avanza la casilla de la ronda y por tanto cambia el turno del jugador. Además, imprime la situación actual de cada jugador en ese momento (francos, comida y riqueza total).

FIN DE RONDA

Aquí englobamos todas las reglas relacionadas con el fin de ronda. En este caso tendrán una comprobación de que la fase 1 y la 2 ya están completada, además de que el jugador ha llegado a la casilla final, la 7. En las de recoger la cosecha será necesario comprobar también si se ha realizado la cosecha de cada jugador con su respectivo atributo y modificarlo al ejecutar la regla. Por otro lado, en las de alimentar a los trabajadores, del mismo modo que el anterior, se comprobará el atributo que indica si se ha pagado la comida o no y se cambiará cuando corresponda.

RecogerCosechaJugador1: en caso de haber cosecha al final de la ronda y cumplir los requisitos el jugador 1 recoge el bien de tipo vaca que le corresponde.

RecogerCosechaJugador2: en caso de haber cosecha al final de la ronda y cumplir los requisitos el jugador 2 recoge el bien de tipo vaca que le corresponde.

AlimentarTieneComidaJugador1: el jugador en turno alimenta al finalizar la ronda pues tiene suficiente comida para pagar.

AlimentarNoTieneComidaPeroSiDineroJugador1: el jugador en turno alimenta al finalizar la ronda, pues no tiene suficiente comida, pero si dinero para pagar.

AlimentarNoTieneComidaNiDineroJugador1: el jugador en turno tiene que pedir un préstamo para alimentar al finalizar la ronda pues no tiene suficiente comida ni dinero para pagar.

AlimentarTieneComidaJugador2: el jugador que no está en turno alimenta al finalizar la ronda pues tiene suficiente comida para pagar.

AlimentarNoTieneComidaPeroSiDineroJugador2: el jugador que no está en turno alimenta al finalizar la ronda pues no tiene suficiente comida, pero sí dinero para pagar.

AlimentarNoTieneComidaNiDineroJugador2: el jugador que no está en turno tiene que pedir un préstamo para alimentar al finalizar la ronda pues no tiene suficiente comida ni dinero para pagar.

FinalizarRonda: finaliza la ronda y pasa a la siguiente. Tendremos que comprobar si ambos jugadores han alimentado (*pagado_comida*) a sus trabajadores. Al ejecutarse quitaremos la instancia de la ronda que se acaba de jugar y a la nueva cambiaremos el atributo *jugando_ronda* a *true*.

FIN DEL JUEGO

En estas últimas reglas será necesario comprobar que la ronda es la última, la 14. Luego se actualizarán las riquezas de caza jugador teniendo en cuenta sus préstamos, se compararán y se decidirá el ganador.

AcabarGanador1: finaliza la ejecución del juego en caso de que ya se haya jugado la última ronda y devuelve como ganador el jugador 1.

AcabarGanador2: finaliza la ejecución del juego en caso de que ya se haya jugado la última ronda y devuelve como ganador el jugador 2.

AcabarEmpate: finaliza la ejecución del juego en caso de que ya se haya jugado la última ronda e informa que ha habido un empate.

3.- MANUAL DE USUARIO

Para utilizar correctamente el código proporcionado el usuario debe tener instalado el interpretador del lenguaje CLIPS. A continuación, para cargar los ficheros, podrá hacerlo haciendo clic en ***"Environment"*** → ***"Load constructs"*** o escribiendo el mismo en la *shell* ***(load <nombre_del_fichero>)***. Los ficheros en cuestión son: ***"ontologia.clp"*** (clases definidas con sus respectivos atributos) y ***"reglas.clp"*** (reglas de inferencia para el correcto funcionamiento del programa).

Una vez hecho esto, deberá cargar de la misma manera su propio fichero de prueba con las instancias y hechos iniciales que desee incluir, siguiendo la estructura de los ficheros de este

tipo que proporcionamos. También puede observar el comportamiento del proyecto con nuestros ficheros de prueba, ***"prueba-X.clp"***, donde X será el número identificador de la prueba.

A continuación, ejecutará los siguientes comandos en la *shell*:

- *(reset)*
- *(run)*

Si se han seguido las instrucciones correctamente, el programa se ejecutará y simulará una partida del juego. Cuando esta termine podrá consultarse el fichero *"salida-prueba-X"*, con las trazas de la prueba realizada, es decir, las acciones y sus resultados en orden que se han ido ejecutando para avanzar a lo largo de las fases del juego.

4.- PRUEBAS REALIZADAS

Las pruebas se harán de manera secuencial, primero comenzaremos con una ejecución simple del juego y a medida que se vayan superando las pruebas aumentaremos la complejidad, añadiendo nuevas funcionalidades y manteniendo las anteriores.

prueba-1.clp

En la primera prueba, realizaremos la ejecución más simple posible. En ella no hay ningún edificio, por lo que la acción principal sólo podrá ser la de tomar bienes de la oferta. Además, no se realiza cosecha al final de la ronda, ni hay intereses en las casillas de las ofertas.

En ella hemos obtenido unos resultados correctos en los que el juego se termina con un ganador. Las riquezas dan valores negativos muy bajos debido a la gran cantidad de préstamos que solicitan los jugadores al no tener fuentes de ingresos más que el obtener bienes. En este caso podemos ver que la riqueza total y los francos son iguales porque no hay edificios que puedan modificar la riqueza, y los préstamos se calculan al finalizar el juego.

prueba-2.clp

En esta segunda prueba, incrementamos la complejidad, añadiendo intereses a las ofertas. Sigue sin haber edificios ni cosecha al final de la ronda.

En este caso, obtenemos unos resultados todavía más bajos, ya que al existir intereses los jugadores tienen que pagar más y, por lo tanto, tienen menos dinero y menos riqueza. Aun así, de nuevo obtenemos un ganador sin problema.

prueba-3.clp

Para la tercera prueba, añadimos cosechas al final de las rondas, pero no edificios. Esto hará que los jugadores posean más bienes.

Al poseer más bienes, quizá en un caso donde antes debían pedir un préstamo, ahora no es necesario, por lo que las riquezas de cada jugador son mejores (más altas).

prueba-4.clp

Ahora aumentamos aún más la complejidad y añadimos edificios sin coste de construcción, es decir, aquellos que se pueden construir sin pagar en bienes.

Esta prueba ya empieza a dar más juego a la partida, ya que ahora se podrán realizar dos acciones diferentes a la hora de elegir la acción principal. Además, se podrán realizar también las acciones opcionales de comprar y vender los edificios. En este caso esperábamos observar que las riquezas aumentarían en mayor medida, ya que se tienen en cuenta también los valores de los edificios adquiridos. Sin embargo, los préstamos están muy presentes a lo largo de la ejecución y eso se ve reflejado en la riqueza. Quizá es debido a la poca variedad de edificios que se han implementado, que no da demasiado margen de maniobra a la hora de realizar acciones.

prueba-5.clp

En la última prueba añadimos los edificios que tienen coste de construcción. Esta sería la ejecución completa, con todas las funcionalidades implementadas.

En esta prueba esperábamos que los jugadores acabarán con las mayores riquezas, pues poseen más tipos de edificios y se benefician de sus acciones. Sin embargo, la cantidad de edificios añadidos no deja ver esta mejora a la larga en las riquezas, pues aunque los jugadores ganan francos cuando otro entra en sus propiedades, también pierden al comprarlos y al entrar en el del otro.

Todas las pruebas finalizan la partida correctamente, modificando cada instancia, atributo y hecho de la manera esperada. Se puede consultar en las trazas de los ficheros *"salida-prueba-X",* donde X es el número identificador de la prueba correspondiente.

Como se puede observar las riquezas de los jugadores para las pruebas realizadas no son muy buenas, esto se debe a que las instancias añadidas cuentan con los datos que deberían tener en caso de que se tratase del juego completo. Sin embargo, al ser el nuestro una simplificación del mismo no se consiguen los resultados esperados. Esto se podría ajustar reduciendo valores iniciales como el de comida a pagar en una determinada ronda, de esta manera se tendrían que pedir menos préstamos y las riquezas totales serían más altas.

5.- CONCLUSIONES

Queremos destacar que nosotros tan solo hemos hecho una aproximación a la solución del juego Le Havre, ya que el objetivo principal no es que la solución se asemeje lo máximo posible al problema, sino aprender a utilizar el lenguaje CLIPS correctamente. Este subproblema implementado es fiel y representativo del problema completo, simplemente se han relajado algunas de las funcionalidades:

1. El juego se ha desarrollado para dos jugadores
2. No se han incluido cartas de barcos

3. Los edificios incluidos pueden tener como acción la obtención de un tipo de bien o permitir al jugador construir otro edificio. A su vez el coste de construcción debe estar relacionado con un único tipo de bien.
4. Los bienes de tipo comida se han limitado al tipo vaca.
5. Cuando no se tienen francos para pagar algo el jugador recurre a pedir un préstamo, no vende uno de sus edificios.
6. Las acciones posibles en el turno final son iguales a las del resto de turnos de la partida.

La estrategia elegida para ejecutar el programa ha sido *random* pues así se aleatoriza la elección de las acciones opcionales y se convierte en una partida mucho más fiel a la realidad. Para las reglas que tienen que ejecutarse sí o sí en un determinado momento se les ha añadido una prioridad con *"salience"*. Estas son las de pagar intereses para el jugador que no está en turno, pues aunque no lo esté debe pagar al añadirse una oferta con intereses. Las de construir, pues así obligamos a tras moverse a un edificio, cuya acción es poder construir, construya uno. La de pedir un préstamo, pues un jugador no puede quedarse con los francos en negativo, así que cuando lo detecta, los pide. Y por último, las de recoger cosecha, para que no se realice la acción de alimentar si no se ha recogido la cosecha en caso de haberla.

6.- COMENTARIOS PERSONALES Y PROBLEMAS

Como comentarios personales, consideramos que la realización de esta práctica te ayuda a fijar los conocimientos adquiridos en la teoría. Es una práctica extensa, además en nuestro caso era la primera vez que trabajamos con el lenguaje CLIPS. Esto nos ocasionó avanzar muy lento y de manera incorrecta al principio de la implementación, cosa que hemos tenido que ir corrigiendo a lo largo de esta fase. Una vez la práctica está más avanzada, se hace más llevadera, pues a parte de conocer mejor el lenguaje de programación, el conocimiento sobre el juego implementado es más amplio.