make.tol de ChRules.RandRecursive

ChRules.RandRecursive es un programa de aplicacion de reglas de reescritura que: a) aplica a un area rectangular de caracteres, b) reglas de transformacion de areas rectangulares de caracteres y c) que juntas forman una base de reglas de transformacion del contenido de ese area con un cierto objetivo. Las reglas de ChRules.RandRecursive son del tipo [condicion, accion], esto es: a) si se cumple la condicion b) entonces se aplica la accion de transformacion. Tanto la parte de la condicion como la de la accion son 2 rectangulos de caracteres, en principio de identicas dimensiones, por ejemplo de 2x3 caracteres, de 1x2 caracteres, 3x5 caracteres, etc. La parte inicial del nombre del programa, ChRules, proviene de estas caracteristicas, Ch de Ch(aracters) y Rules de reglas, esto es, que se podrian llamar reglas de caracteres. La idea basica del funcionamiento es la siguiente: a) si en el estado actual del area de caracteres existe algun subarea rectangular con el mismo contenido que la parte de condicion de una regla, b) entonces dicha regla es aplicable y de aplicarse el subarea rectangular del area de caracteres que coincide con la condicion es sobreescrita, conservando la forma, con el area rectangular de caracteres de la accion de la regla. Por tanto, estas reglas de rectangulos de caracteres que utiliza el programa ChRules.RandRecursive pueden considerarse como reglas de reescritura, pero, a diferencia de otras reglas de reescritura, en vez de trabajar con secuencias de caracteres trabajan con areas rectangulares de caracteres.

Una caracteristica particular del programa ChRules.RandRecursive es que, en lenguaje Tol, para la programacion de las funciones como EvalSet(), For(), Select(), Classify(), Sort(), etc. existen 2 formas de hacerlo: a) La primera y mas habitual es declarar el codigo a evaluar dentro del propio parametro de tipo codigo. Esto es, si es por ejemplo, un EvalSet(conjunto, codigo) entonces se programa el codigo, dentro de la propia llamada, como una funcion sin nombre, por ejemplo, como EvalSet(coorRC, Set(Set rc) { ...codigo... }); b) La segunda forma, mucho menos frecuente, es declarar primero la funcion que hay que realizar y, despues, llamar a la funcion que la invoca. Esta forma tiene mucho sentido cuando a la funcion que hay que realizar se la va a invocar desde varias sentencias. De esta forma, por ejemplo, se declara primero las funciones, matchRC(...parametro....) { codigo } o matchWidth(...parametro....) { codigo } y luego se invoca directamente a esa funcion dentro del EvalSet(), por ejemplo, EvalSet(coorRC, matchRC). Esta 2ª forma es mas infrecuente en Time Oriented Programming. A diferencia de otros programas Tol, en ChRules.RandRecursive se emplean ambos estilos de programacion de forma indistinta. Las versiones iniciales de este programa permitieron evaluar las primeras versiones de Tol por lo que, todavia hoy, ChRules.RandRecursive funciona en muchas versiones de Tol como las 1.1.1, 1.1.5, 1.1.6 y 2.0.1. y conserva en su estilo de programacion caracteristicas muy primigenias.

El programa ChRules.RandRecursive tiene 2 particularidades que le definen y que forman parte de su nombre, RandRecursive, que son: a) El ciclo del motor de comprobacion y aplicacion de reglas es recursivo. b) Tanto la aplicacion de reglas como la seleccion de subareas donde aplicar la regla son escogidas al azar. La aplicacion al azar de las reglas significa que: a) Si en un determinado ciclo de evaluacion y con un determinado estado del area rectangular de caracteres son aplicables varias de las reglas de la base de reglas por conincidir su parte de condicion, al menos, con un subarea del area, se elige y se aplica al azar una de esas reglas. b) A su vez, al aplicar una regla, si su parte de condicion realiza

match con mas de un subarea rectangular del area de caracteres, entonces se elige un subarea al azar y es sobre ese subarea sobre la que se aplica la transformacion. Esto implica una doble aleatoriedad en reglas y subareas de aplicacion lo que hace que cada uno de los casos de ejemplo que se incluyen en este programa y a los que se aplica el motor de reglas de ChRules.RandRecursive puede evolucionar, en cada ejecucion, de una manera muy diferente.

ChRules.RandRecursive incluye un conjunto de pares [area, base de reglas] con diferentes caracteristicas y objetivos: rnd) Es para comprobar que el comportamiento es realmente aleatorio tanto en la aplicacion de las reglas como sobre las subareas de aplicacion. cua) Es una base de reglas constructivas, realizan un crecimiento aleatorio cuadriculando su area de aplicacion, termina cuando todo el area esta cuadriculada. cel) Es un automata celular donde un conjunto de celulas se mueven de forma libre por el area, reproduciendose por parejas y engendrando entre 2 una nueva celula y tambien pueden morir cuando estan demasiado juntas (superpoblacion) o, por el contrario, demasiado aisladas. Es, por tanto, una base de reglas de movimiento, de creacion y de destruccion como la vida misma. bat) Es una batalla fundamentalmente destructiva donde 2 bandos de bombarderos y lanzaderas de misiles se enfrentan de una forma equilibrada. Esto es, todas las reglas a favor o en contra de uno de ellos tienen sus reglas equivalentes a favor o en contra del otro. Adicionalmente, incluye pequeñas tactivas defensivas como contrarrestar con una bomba un misil del contrario, contrarrestar con un misil una bomba del contrario, cerrar las defensas para evitar un impacto y , ciertas complicaciones, como por ejemplo, que bombas y misiles pueden sufrir desviaciones y que con al menos una defensa averiada los bombarderos y las lanzaderas no pueden ni cerrarse ni moverse. wal) Es una base de reglas principalmente destructiva, donde una serie de lanzaderas de misiles tratan de derribar un muro en el que a su vez los elementos son explosivos y al recibir un impacto pueden producir cadenas de destruccion que se transmiten por los elementos adyacentes. A diferencia de las anteriores esta base de reglas incluye una regla de deteccion del final del proceso de deteccion de reglas.

ChRules.RandRecursive visualiza sus resultados de 2 formas diferentes: a) Mediante una traza de evolucion del mapa por pantalla. b) Mediante una traza en un fichero en disco, escrita en Javascript, que permite la posterior simulacion de los resultados. Esta traza en Javascript necesita ser retocada, por ejemplo, en la finalizacion de los arrays, para poder ser empleada por un simulador Javascript. El programa se estructura en base a un probrama principal make.tol que se incluye tantos ficheros de aplicaciones como casos se han programado. Cada uno de los casos consta de: a) Un area de caracteres inicial que puede ser cosiderado como el mapa de operaciones o la base de hechos. b) Una base de reglas con las reglas de caracteres que operan sobre dicho mapa. Las funciones principales de ChRules.RandRecursive son: a) Set EngineCicle() que es el motor recursivo de aplicacion de reglas. b) Set EngineGetRule() que elige al azar una regla aplicable y para ello se apoya en EngineMatch() que encuentra los posibles match entre subareas del area y la parte de condicion de las reglas. c) Set EngineApplyRule() que aplica una regla y transforma un subarea al azar, de entre las transformables, del area y para ello se apoya en la funcion EngineApplyAction() que aplica una accion a un area. d) Real PackPrint() que se encarga de visualizar las reglas, las areas en su estado inicial y en su evolucion e, incluso, de la creacion d e las trazas, como arrays en Javascript, que permitiran la simulacion. Aunque escribe Javascript, este programa no es categorizado como de metaprogramacion pues el codigo Javascript que genera no es 100% funcional.

Árbol de ficheros

ChRules.RandRecursive programa de aplicacion de reglas de rectangulos de caracteres

- ← make.tol aplica unas bases de reglas de reescritura a varios escenarios
- ← make.bat mandato de ejecucion del programa de aplicacion de reglas
- tol directorios que contienen fichero de codigo fuente Tol
  - app directorio con areas, bases de hechos, y con bases de reglas
    - ← rnd.tol test del comportamiento aleatorio del motor recursivo
    - ← cua.tol base de reglas para crecimiento cuadriaculando un area
    - ← cel.tol automata celular de movimiento, reproduccion y muerte
    - ← bat.tol batalla entre 2 frentes con pequeñas tacticas defensivas
    - ← wal.tol base de reglas fundamentalmete destructiva de su area
  - ← inc.tol para la inclusion de todas las bases de reglas del programa
- simulator directorio del simulador del motor de reglas en Javascript
  - css directorio para css, Cascading Style Sheets, del simulador
    - ← simulator.css css para simular areas de aplicacion de las reglas
  - src directorio de codigo fuente Javascript del simulador de reglas
    - ← simulator.js simula el funcionamiento del motor de aplicacion de reglas
    - ← simulatorarray.js array con ejemplos de evolucion para cada base de reglas
- → startlog.txt log Tol de lectura de reglas y evolucion del automata celular
- → traceseg.txt traza de evolucion del automata celular casi en Javasript
- → simulator.html simulador del motor recursivo de reglas de areas de caracteres
- → chrules_randrecursive.pdf funciones del motor de aplicacion de reglas de caracteres

Declaraciones

Constantes

- `Text TrcFil`
  Fichero de traza.

Funciones: Generales

- `Set GetRand(Set setInp)`
  Retorna un conjunto al azar de los que formal el conjunto de entrada setInp. Si setInp es Empty retorna Empty.
- `Set Build2DSet(Set txtSet, Real iniPos, Real endPos)`

Retorna un conjunto, Set, de 2 dimensiones a partir de un conjunto de textos de 1 dimension. En el conjunto de entrada cada elemento es considerado una linea y se separan los caracteres de 1 en 1. Los parametros iniPos y endPos indican la porcion de las lineas a considerar en esta separacion, ambos inclusive. Estos parametros deben ser los correctos pues, por ejemplo, esta funcion no comprueba que no superen la longitud de la linea mas corta. Ejemplo: ini=3 end=7 | | [['123456789', => [[[['3','4','5','6','7']], 'aaabbbccc', [['a','b','b','b','c']], 'AABBCCDDE']] [['B','B','C','C','D']]]]

- `Real PackPrint(Set chrTab, Real trcCtr)`
  Visualiza un conjunto, Set, de textos de 2 dimensiones por pantalla. Para ello efectua una operacion inversa a la que realiza la funcion Build2DSet() pasando a una sola linea empaquetada todos los elementos de cada fila del conjunto tabla chrTab. Al final, pone una linea de separacion tras la impresion del area. Dependiendo del valor de trcCtr, control de traza realiza: - si 1 inicializa un array en un fichero, - si >= 2 añade un texto al array y - si <= 0 solo pantalla.

Funciones: Area

- `Real AreaHeight(Set area)`
  Retorna el alto de un area.
- `Real AreaWidth(Set area)`
  Retorna el alto de un area.
- `Set AreaBuild(Set areBdy)`
  Construye y retorna un area, areSet, con su ancho determinado por el ancho del texto de su primera linea, areWid. Como efecto lateral esta funcion visualiza informacion sobre el area.

Funciones: Reglas

- `Text RuleArrow`
  El simbolo de inferencia para las reglas.
- `Set RulePattern(Set rule)`
  Retorna el area del patron de una regla.
- `Set RuleAction(Set rule)`
  Retorna el area de la accion de una regla.
- `Real RuleHeight(Set rule)`
  Retorna el alto del patron de una regla.
- `Real RuleWidth(Set rule)`
  Retorna el ancho del patron de una regla.
- `Set RuleBuild(Set rulBdy)`
  A partir de una regla en forma de texto, rulTxt, construye y retorna una regla como un conjunto, calculando el tamaño de las areas de la regla a partir del simbolo de inferencia (el que se declare en cada caso, por ejemplo, =>). Busca el simbolo de inferencia en la primera linea de texto de la regla y con su posicion se calcula el ancho de la regla, rulWid. Si no se encuentra el simbolo de inferencia o aparece en una posicion absurda emite un mensaje de error y retorna el conjunto vacio. La regla debe estar bien formada, por ejemplo: ruleBody = SetOfText('.X.=>Z.Z', 'X.X .Z.'); Como efecto lateral esta funcion visualiza informacion sobre la regla.

Funciones: Motor de reglas

- `Set EngineMatch(Set area, Set rule)`
  Retorna un conjunto, set, con las coordenadas (r,c) de fila (row) y columna (column) donde el patron de la regla (rule) equipara (match) con el area. Si no hay equiparacion posible, retorna las coordenadas (0,0). La equiparacion se hace caracter a caracter. Ha de hacerse notar que esta funcion encuentra todas las equiparaciones posibles, pero solo retorna la primera. Esto hace que sea no muy eficiente. La funcion esta construida en base a una funcion elementMatch() que retorna cierto si hay match en una celda concreta del patron, esta funcion es utilizada por matchWidth() que retorna cierto si hay correspondencia a lo largo de una fila del patron, sobre ella trabaja matchRC() que retorna una terna (Set) con la fila R, la columna C y cierto si todo el patron equipara a partir de las coordenadas (r,c) del area y falso si no equipara. Notese como el estilo de programacion que se utiliza para las funciones EvalSet() es declarar primero la funcion, por ejemplo, matchRC() o matchWidth() y luego invocar directamente a la funcion dentro del EvalSet(). EvalSet(coorRC, matchRC); en vez de EvalSet(coorRC, Set (Set rc) { ...codigo... });
- `Set EngineApplyAction(Set area, Set rule, Set matchRC)`
  Retorna un conjunto, Set, resultado de aplicar al area la regla (rule) en las coordenadas (r,c) que indica matchRC. Notese el estilo utilizado en las funciones EvalSet() para las que se declara primero la funcion que luego se invocar dentro del EvalSet().
  EvalSet(rangeHeight, buildLine); en vez de EvalSet(rangeHeight, Set(Real rCount) { ...codigo... });
- `Real EngineMatchOk(Set rc)`
  Retorna cierto si hay match en las coordenadas (r,c), esto es, si ambas son mayor que cero.
- `Set EngineGetRule(Set area, Set ruleBase)`
  Retorna una regla al azar de ruleBase aplicable al area. Si ninguna se puede aplicar retorna el conjunto vacio, para la funcion que llama esto deberia significar que el sistema ha terminado, pues no hay reglas que aplicar.
- `Set EngineApplyRule(Set area, Set rule)`
  Retorna un nuevo area, que es conjunto, resultado de aplicar una regla a un area de entrada. Si esta regla no es aplicable (esto es, no hace match en ninguna coordenada del area), entonces retorna el mismo area de entrada sin transformar. Si bien, tal y como esta programado el motor de aplicacion de reglas, esto no tendria que ocurrir. Si la regla es aplicable en diversas zonas del area la funcion EngineMatch() elegira una area al azar.
- `Set EngineCicle(Set area, Set ruleBase)`
  Ciclo interno del motor de aplicacion de reglas. Aqui es donde se fija la estrategia de aplicacion de reglas, el algoritmo es el siguiente: - Elegir una regla al azar de ruleBase aplicable al area. - Si no hay ninguna regla aplicable entonces todo ha terminado, retorna area. - Si hay al menos una regla aplicable entonces - aplicar la regla al area, - obtener un nuevo area transformada de la anterior y - entrar en recursion con la nueva area y el mismo conjunto de reglas. Se trata, por tanto, de la funcion en la que se realiza la recursion.

Inclusiones

- `Set allInc`
  Inclusion de areas y bases de reglas.

Pruebas

- `Text tstCmd`

  Mandatos de simulacion: rnd, cua, cel, bat, wal.

- `Real tstExe`

  Aplica las reglas al area dependiendo de tstCmd.

Constantes

Text TrcFil

```
////////////////////////////////////////////////////////////////////////////
Text TrcFil = "trace.txt";
////////////////////////////////////////////////////////////////////////////
PutDescription("Fichero de traza.", TrcFil);
////////////////////////////////////////////////////////////////////////////
```

Set GetRand()

```
////////////////////////////////////////////////////////////////////////////
Set GetRand(Set setInp)
////////////////////////////////////////////////////////////////////////////
{
  Real setCrd = Card(setInp);
  If(LE(setCrd, 0), Empty,                                  // El conjunto vacio
     setInp[Min(setCrd, Max(1, Round(Rand(0, setCrd)+0.5)))]) // Set al azar
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto al azar de los que formal el conjunto de entrada setInp.
Si setInp es Empty retorna Empty.",
GetRand);
////////////////////////////////////////////////////////////////////////////
```

Set Build2DSet()

```
////////////////////////////////////////////////////////////////////////////
Set Build2DSet(Set  txtSet, // Conjunto de textos
               Real iniPos, // Posicion inicial para cortar
               Real endPos) // Posicion final para cortar
////////////////////////////////////////////////////////////////////////////
{
  EvalSet(txtSet, Set(Text txtLin)
  {
    For(iniPos, endPos, Text(Real numPos) { Sub(txtLin, numPos, numPos) })
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto, Set, de 2 dimensiones a partir de un conjunto de textos
de 1 dimension.
En el conjunto de entrada cada elemento es considerado una linea y se separan
los caracteres de 1 en 1.
Los parametros iniPos y endPos indican la porcion de las lineas a considerar
en esta separacion, ambos inclusive.
Estos parametros deben ser los correctos pues, por ejemplo, esta funcion no
comprueba que no superen la longitud de la linea mas corta.
Ejemplo:
    ini=3   end=7
        |   |
  [[ '123456789',   => [[ [[ '3','4','5','6','7' ]],
```

```
     'aaabbbccc',          [[ 'a','b','b','b','c' ]],
     'AABBCCDDE' ]]        [[ 'B','B','C','C','D' ]] ]]",
Build2DSet);
////////////////////////////////////////////////////////////////////////////
```

Real PackPrint()

```
////////////////////////////////////////////////////////////////////////////
Real PackPrint(Set  chrTab, // Conjunto de conjuntos de 1 caracter
               Real trcCtr) // Control de traza
////////////////////////////////////////////////////////////////////////////
{
//Real System("cls"); Limpiar la pantalla antes de imprimir
  Set linSet = EvalSet(chrTab, Text(Set rowChr)
  {
    Text txtLin = BinGroup("+", rowChr); // En 1 linea el Set de caracteres
    Text WriteLn(txtLin);                // Visualizarlo
    txtLin+";"                           // ; separador no usado en reglas
  });
  Text WriteLn(Repeat("_", 78));         // Poner un separador

  Text trcLin = "  "+Char(34)+SetSum(linSet)+Char(34)+",\n"; // Comillas 34
  Text trcSep = "\n"+Repeat("/", 78)+"\n\n"; // Separador de arrays
  Text trcIni = trcSep + "var trcLog = new Array(\n"; // Array Javascript

  Real If(trcCtr == 1, { Text AppendFile(TrcFil, trcIni); TRUE }, FALSE);
  Real If(trcCtr >= 1, { Text AppendFile(TrcFil, trcLin); TRUE }, FALSE);

  Real Card(linSet)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Visualiza un conjunto, Set, de textos de 2 dimensiones por pantalla.
Para ello efectua una operacion inversa a la que realiza la funcion
Build2DSet() pasando a una sola linea empaquetada todos los elementos de cada
fila del conjunto tabla chrTab.
Al final, pone una linea de separacion tras la impresion del area.
Dependiendo del valor de trcCtr, control de traza realiza:
- si 1 inicializa un array en un fichero,
- si >= 2 añade un texto al array y
- si <= 0 solo pantalla.",
PackPrint);
////////////////////////////////////////////////////////////////////////////
```

Real AreaHeight()

```
////////////////////////////////////////////////////////////////////////////
Real AreaHeight(Set area)
////////////////////////////////////////////////////////////////////////////
{ Card(area) };
////////////////////////////////////////////////////////////////////////////
PutDescription("Retorna el alto de un area.",AreaHeight);
////////////////////////////////////////////////////////////////////////////
```

Real AreaWidth()

```
////////////////////////////////////////////////////////////////////////////
Real AreaWidth (Set area)
////////////////////////////////////////////////////////////////////////////
{ Card(area[1]) };
////////////////////////////////////////////////////////////////////////////
PutDescription("Retorna el alto de un area.",AreaWidth);
////////////////////////////////////////////////////////////////////////////
```

Set AreaBuild()

```
////////////////////////////////////////////////////////////////////////
Set AreaBuild(Set areBdy)
////////////////////////////////////////////////////////////////////////
{
  Real areWid = TextLength(areBdy[1]); // Asume el ancho de la primera linea
  Set  areSet = Build2DSet(areBdy, 1, areWid); // Convierte todo el anchp
  Real numLin = PackPrint(areSet, 0);          // Visualiza por pantalla
  Text WriteLn(FormatReal(numLin, "%.0lf") + " rows");
  Set  areSet
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Construye y retorna un area, areSet, con su ancho determinado por el ancho
del texto de su primera linea, areWid.
Como efecto lateral esta funcion visualiza informacion sobre el area.",
AreaBuild);
////////////////////////////////////////////////////////////////////////
```

Text RuleArrow

```
////////////////////////////////////////////////////////////////////////
Text RuleArrow = " => ";
////////////////////////////////////////////////////////////////////////
PutDescription("El simbolo de inferencia para las reglas.",RuleArrow);
////////////////////////////////////////////////////////////////////////
```

Set RulePattern()

```
////////////////////////////////////////////////////////////////////////
Set RulePattern(Set rule)
////////////////////////////////////////////////////////////////////////
{ rule[1] };
////////////////////////////////////////////////////////////////////////
PutDescription("Retorna el area del patron de una regla.",RulePattern);
////////////////////////////////////////////////////////////////////////
```

Set RuleAction()

```
////////////////////////////////////////////////////////////////////////
Set RuleAction(Set rule)
////////////////////////////////////////////////////////////////////////
{ rule[2] };
////////////////////////////////////////////////////////////////////////
PutDescription("Retorna el area de la accion de una regla.",RuleAction);
////////////////////////////////////////////////////////////////////////
```

Real RuleHeight()

```
////////////////////////////////////////////////////////////////////////
Real RuleHeight(Set rule)
////////////////////////////////////////////////////////////////////////
{ Card(RulePattern(rule)) };
////////////////////////////////////////////////////////////////////////
PutDescription("Retorna el alto del patron de una regla.",RuleHeight);
////////////////////////////////////////////////////////////////////////
```

Real RuleWidth()

```
////////////////////////////////////////////////////////////////////////////
Real RuleWidth(Set rule)
////////////////////////////////////////////////////////////////////////////
{ Card(RulePattern(rule)[1]) };
////////////////////////////////////////////////////////////////////////////
PutDescription("Retorna el ancho del patron de una regla.",RuleWidth);
////////////////////////////////////////////////////////////////////////////
```

Set RuleBuild()

```
////////////////////////////////////////////////////////////////////////////
Set RuleBuild(Set rulBdy) // Regla en forma toda de texto
////////////////////////////////////////////////////////////////////////////
{
  Real rulWid = TextFind(rulBdy[1], RuleArrow) - 1; // Ancho de la regla
  Real bdyLen = TextLength(rulBdy[1]); // Ancho total de la regla
  Real arrLen = TextLength(RuleArrow); // Ancho del simbolo de inferencia

  If(Or(rulWid < 1, arrLen < 0), // Error en el simbolo de inferencia
  {
    Text WriteLn("Regla mal formada");
    Empty
  },
  {               // Pattern del 1 hasta => y action desde => al final
    Set  rulPat = Build2DSet(rulBdy, 1,                rulWid); // Patron
    Set  rulAct = Build2DSet(rulBdy, arrLen+1+rulWid, bdyLen); // Accion

    Text WriteLn("Rule pattern");
    Real PackPrint(rulPat, 0);    // Visualiza por pantalla
    Text WriteLn("Rule action");
    Real PackPrint(rulAct, 0);    // Visualiza por pantalla

    SetOfSet(rulPat, rulAct) // Retorna la regla como un par [patron,accion]
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"A partir de una regla en forma de texto, rulTxt, construye y retorna una
regla como un conjunto, calculando el tamaño de las areas de la regla a partir
del  simbolo de inferencia (el que se declare en cada caso, por ejemplo, =>).
Busca el simbolo de inferencia en la primera linea de texto de la regla y con
su posicion se calcula el ancho de la regla, rulWid.
Si no se encuentra el simbolo de inferencia o aparece en una posicion absurda
emite un mensaje de error y retorna el conjunto vacio.
La regla debe estar bien formada, por ejemplo:
  ruleBody = SetOfText('.X.=>Z.Z',
                       'X.X  .Z.');
Como efecto lateral esta funcion visualiza informacion sobre la regla.",
RuleBuild);
////////////////////////////////////////////////////////////////////////////
```

Set EngineMatch()

```
////////////////////////////////////////////////////////////////////////////
Set EngineMatch(Set area,
                Set rule)
////////////////////////////////////////////////////////////////////////////
{
  Set  pattern       = RulePattern(rule);
  Real patternHeight = RuleHeight(rule);
  Real patternWidth  = RuleWidth(rule);

  Real areaHeight = AreaHeight(area);
  Real areaWidth  = AreaWidth(area);
```

```
  Set matchRC(Set rc) // Funcion
  {
    Real r = rc[1]; // Row
    Real c = rc[2]; // Column

    Real matchWidth(Real dr) // Funcion
    {
      Real elementMatch(Real dc) { area[r+dr-1][c+dc-1] == pattern[dr][dc] };

      Set  rangeWidth = Range(1,patternWidth,1);
      Set  matchWidth = EvalSet(rangeWidth,elementMatch);
      BinGroup("*",matchWidth)
    };

    Set rangeHeight = Range(1,patternHeight,1);

    Set matchHeight = EvalSet(rangeHeight, matchWidth);

    SetOfReal(r,c,BinGroup("*",matchHeight))
  };

  Set rangeR = Range(1, 1 + areaHeight - patternHeight, 1);
  Set rangeC = Range(1, 1 + areaWidth  - patternWidth,  1);
  Set coorRC = CartProd(rangeR, rangeC);

  Set match  = EvalSet(coorRC, matchRC);

  Set matchTrue = Select(match, Real(Set s){s[3]});
  Real last  = Card(matchTrue);

  If(! last, SetOfReal(0,0),
  {
    Set mthRnd =  GetRand(matchTrue); // Extraer un area al azar
    [[ mthRnd[1], mthRnd[2] ]] // Retornar las coordenadas de match
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto, set, con las coordenadas (r,c) de fila (row) y columna
(column) donde el patron de la regla (rule) equipara (match) con el area.
Si no hay equiparacion posible, retorna las coordenadas (0,0).
La equiparacion se hace caracter a caracter.
Ha de hacerse notar que esta funcion encuentra todas las equiparaciones
posibles, pero solo retorna la primera.
Esto hace que sea no muy eficiente.
La funcion esta construida en base a una funcion elementMatch() que retorna
cierto si hay match en una celda concreta del patron, esta funcion es
utilizada por matchWidth() que retorna cierto si hay correspondencia a lo
largo de una fila del patron, sobre ella trabaja matchRC() que retorna una
terna (Set) con la fila R, la columna C y cierto si todo el patron equipara a
partir de las coordenadas (r,c) del area y falso si no equipara.
Notese como el estilo de programacion que se utiliza para las funciones
EvalSet() es declarar primero la funcion, por ejemplo, matchRC() o
matchWidth() y luego invocar directamente a la funcion dentro del EvalSet().
EvalSet(coorRC, matchRC);
en vez de
EvalSet(coorRC, Set (Set rc) { ...codigo... });",
EngineMatch);
////////////////////////////////////////////////////////////////////////////
```

Set EngineApplyAction()

```
////////////////////////////////////////////////////////////////////////////
Set EngineApplyAction(Set area,
                      Set rule,
                      Set matchRC)
////////////////////////////////////////////////////////////////////////////
{
  Set  action       = RuleAction(rule);
  Real actionHeight = RuleHeight(rule);
  Real actionWidth  = RuleWidth(rule);
```

```
  Real areaHeight = AreaHeight(area);
  Real areaWidth  = AreaWidth(area);

  Real r = matchRC[1]; // Row
  Real c = matchRC[2]; // Column

  Set buildLine(Real rCount)
  {
    Text pointXY(Real cCount)
    {
      Real inside = And(rCount>=r, rCount<Real(r+actionHeight),
                        cCount>=c, cCount<Real(c+actionWidth));

      Text If(inside,
              action[1+rCount-r][1+cCount-c],
              area[rCount][cCount])
    };
    Set  rangeWidth = Range(1,areaWidth,1);

    EvalSet(rangeWidth, pointXY)
  };
  Set rangeHeight = Range(1, areaHeight, 1);

  EvalSet(rangeHeight, buildLine)
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un conjunto, Set, resultado de aplicar al area la regla (rule) en
las coordenadas (r,c) que indica matchRC.
Notese el estilo utilizado en las funciones EvalSet() para las que se declara
primero la funcion que luego se invocar dentro del EvalSet().
EvalSet(rangeHeight, buildLine);
en vez de
EvalSet(rangeHeight, Set(Real rCount) { ...codigo... });",
EngineApplyAction);
////////////////////////////////////////////////////////////////////////
```

Real EngineMatchOk()

```
////////////////////////////////////////////////////////////////////////
Real EngineMatchOk(Set rc)
////////////////////////////////////////////////////////////////////////
{ And(Real(rc[1])>0,Real(rc[2])>0) };
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna cierto si hay match en las coordenadas (r,c), esto es, si ambas son
mayor que cero.",
EngineMatchOk);
////////////////////////////////////////////////////////////////////////
```

Set EngineGetRule()

```
////////////////////////////////////////////////////////////////////////
Set EngineGetRule(Set area,
                  Set ruleBase)
////////////////////////////////////////////////////////////////////////
{
  Set selRul = Select(ruleBase, Real(Set rule)
  {
    Set matchRC = EngineMatch(area,rule); // Coordenadas en que hace match
    EngineMatchOk(matchRC)                // Retorna cierto si hace match
  });
  GetRand(selRul) // De entre todas las que hacen match retorna una al azar
};
////////////////////////////////////////////////////////////////////////
```

```
PutDescription(
"Retorna una regla al azar de ruleBase aplicable al area.
Si ninguna se puede aplicar retorna el conjunto vacio,
para la funcion que llama esto deberia significar que el sistema ha terminado,
pues no hay reglas que aplicar.",
EngineGetRule);
////////////////////////////////////////////////////////////////////////////
```

Set EngineApplyRule()

```
////////////////////////////////////////////////////////////////////////////
Set EngineApplyRule(Set area,
                    Set rule)
////////////////////////////////////////////////////////////////////////////
{
  Set matchRC = EngineMatch(area,rule);
  If(!EngineMatchOk(matchRC), area,
                              EngineApplyAction(area,rule,matchRC))
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un nuevo area, que es conjunto, resultado de aplicar una regla a
un area de entrada.
Si esta regla no es aplicable (esto es, no hace match en ninguna coordenada
del area), entonces retorna el mismo area de entrada sin transformar.
Si bien, tal y como esta programado el motor de aplicacion de reglas, esto
no tendria que ocurrir.
Si la regla es aplicable en diversas zonas del area la funcion EngineMatch()
elegira una area al azar.",
EngineApplyRule);
////////////////////////////////////////////////////////////////////////////
```

Set EngineCicle()

```
////////////////////////////////////////////////////////////////////////////
Set EngineCicle(Set  area,
                Set  ruleBase)
////////////////////////////////////////////////////////////////////////////
{
  Set selRul = EngineGetRule(area, ruleBase);    // Regla aplicable al azar
  If(!Card(selRul), area, // No hay reglas que aplicar, es el fin
  {
    Set  newAre = EngineApplyRule(area, selRul); // Transforma el area
    Real PackPrint(newAre, 2);                   // Visualiza el area
    EngineCicle(newAre, ruleBase)                // Recursion
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Ciclo interno del motor de aplicacion de reglas.
Aqui es donde se fija la estrategia de aplicacion de reglas,
el algoritmo es el siguiente:
- Elegir una regla al azar de ruleBase aplicable al area.
- Si no hay ninguna regla aplicable entonces todo ha terminado, retorna area.
- Si hay al menos una regla aplicable entonces
  - aplicar la regla al area,
  - obtener un nuevo area transformada de la anterior y
  - entrar en recursion con la nueva area y el mismo conjunto de reglas.
Se trata, por tanto, de la funcion en la que se realiza la recursion.",
EngineCicle);
////////////////////////////////////////////////////////////////////////////
```

Set allInc

```
////////////////////////////////////////////////////////////////////////////
Set  allInc = Include("tol/inc.tol");
```

```
////////////////////////////////////////////////////////////////////////////
PutDescription("Inclusion de areas y bases de reglas.", allInc);
////////////////////////////////////////////////////////////////////////////
```

Text tstCmd

```
////////////////////////////////////////////////////////////////////////////
Text tstCmd = "cel"; // Caso de simulacion
////////////////////////////////////////////////////////////////////////////
PutDescription("Mandatos de simulacion: rnd, cua, cel, bat, wal.", tstCmd);
////////////////////////////////////////////////////////////////////////////
```

Real tstExe

```
////////////////////////////////////////////////////////////////////////////
Real tstExe = Case(
  tstCmd == "rnd", { PackPrint(RndAre, 1); // Prueba ejecucion aleatoria
                     Card(EngineCicle(RndAre, RndRul)) },

  tstCmd == "cua", { PackPrint(CuaAre, 1); // Crecimiento cuadriculando area
                     Card(EngineCicle(CuaAre, CuaRul)) },

  tstCmd == "cel", { PackPrint(CelAre, 1); // Automata celular
                     Card(EngineCicle(CelAre, CelRul)) },

  tstCmd == "bat", { PackPrint(BatAre, 1); // Batalla de bombas y misiles
                     Card(EngineCicle(BatAre, BatRul)) },

  tstCmd == "wal", { PackPrint(WalAre, 1); // Derribando un muro
                     Card(EngineCicle(WalAre, WalRul)) },

  TRUE,            FALSE); // No hace nada
////////////////////////////////////////////////////////////////////////////
PutDescription("Aplica las reglas al area dependiendo de tstCmd.", tstExe);
////////////////////////////////////////////////////////////////////////////
```

rnd.tol de ChRules.RandRecursive

Base de reglas y area para probar el comportamiento aleatorio del motor recursivo y aleatorios de reglas de rectangulos de caracteres. Permite comprobar que el comportamiento es realmente aleatorio tanto en la aplicacion de las reglas como sobre las subareas de aplicacion.

Declaraciones

Constantes

- `Set RndAre`
  Area para probar el comportamiento aleatorio.
- `Set RndRul`
  Base de reglas para probar el comportamiento aleatorio.

Constantes

Set RndAre

```
////////////////////////////////////////////////////////////////////////////
Set RndAre = AreaBuild(
[["--------",
  "-00--00-",
  "-00--00-",
  "--------",
  "-00--00-",
  "-00--00-",
  "--------"]]);
////////////////////////////////////////////////////////////////////////////
PutDescription("Area para probar el comportamiento aleatorio.", RndAre);
////////////////////////////////////////////////////////////////////////////
```

Set RndRul

```
////////////////////////////////////////////////////////////////////////////
Set RndRul = SetOfSet(
  RuleBuild([["00 => 11",
              "00    11"]]),   // De cero a uno

  RuleBuild([["00 => 22",
              "00    22"]]),   // De cero a dos

  RuleBuild([["00 => 33",
              "00    33"]]),   // De cero a tres

  RuleBuild([["00 => 44",
              "00    44"]]));  // De cero a tres
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Base de reglas para probar el comportamiento aleatorio.",
RndAre);
////////////////////////////////////////////////////////////////////////////
```

cua.tol de ChRules.RandRecursive

Base de reglas y area para crecimiento cuadriaculando un area. Esta base de reglas es constructiva, sus reglas realizan un crecimiento aleatorio cuadriculando su area de aplicacion, termina cuando todo el area esta cuadriculada.

Declaraciones

Constantes

- `Set CuaAre`
  Area para crecimiento cuadriaculando un area.
- `Set CuaRul`
  Base de reglas para crecimiento cuadriaculando un area.

Constantes

Set CuaAre

```
////////////////////////////////////////////////////////////////////////
Set CuaAre = AreaBuild(
[[".............",
  ".............",
  ".............",
  ".............",
  "....+........",
  ".............",
  ".............",
  ".............",
  "............."]]);
////////////////////////////////////////////////////////////////////////
PutDescription("Area para crecimiento cuadriaculando un area.", CuaAre);
////////////////////////////////////////////////////////////////////////
```

Set CuaRul

```
////////////////////////////////////////////////////////////////////////
Set CuaRul = SetOfSet(
  RuleBuild([["+· => +-"]]), // Brazo hacia la derecha

  RuleBuild([["·+ => -+"]]), // Brazo hacia la izquierda

  RuleBuild([["+ => +",
              "·    |"]]),   // Brazo hacia abajo

  RuleBuild([["· => |",
              "+    +"]]),   // Brazo hacia arriba

  RuleBuild([["-· => -+"]]), // Remate hacia la derecha

  RuleBuild([["·- => +-"]]), // Remate hacia la izquierda

  RuleBuild([["| => |",
              "·    +"]]),   // Remate hacia abajo

  RuleBuild([["· => +",
              "|    |"]]));  // Remate hacia arriba
////////////////////////////////////////////////////////////////////////
PutDescription(
"Base de reglas para crecimiento cuadriaculando un area.",
CuaRul);
////////////////////////////////////////////////////////////////////////
```

cel.tol de ChRules.RandRecursive

Base de reglas y area de un automata celular que simula un juego de la vida. Los automatas celulares son modelos de sistemas dinamicos que evoluciona en forma de pasos discretos y que pueden modelizar sistemas naturales basados en un conjunto, mas o menos grande, de seres simples que interactuan unos con otros de forma local, por cercania. En la teoria, en la decada de los 40, los automatas celulares fueron plateados por Konrad Zuse y Stanislaw Ulam. En la practica, en los años 50 y dentro de la informatica, el concepto de automata celular fue planteado por John von Neumann. // Esta base de reglas emula a un automata celular donde un conjunto de celulas: a) se mueven de forma libre por el area, b) reproduciendose por parejas y engendrando entre 2 una nueva celula y c) tambien pueden morir cuando: - estan demasiado juntas (superpoblacion) o, por el contrario, - demasiado aisladas.

Declaraciones

Constantes

- `Set CelAre`
  Area para un automata celular.
- `Set CelRul`
  Base de reglas para probar el comportamiento de un automata celular.

Constantes

Set CelAre

```
////////////////////////////////////////////////////////////////////////////
Set CelAre = AreaBuild(
[["··············",
  "··············",
  "··············",
  "·····O·O·····",
  "····O··O··O····",
  "·····O·O·····",
  "··············",
  "··············",
  "··············"]]);
////////////////////////////////////////////////////////////////////////////
PutDescription("Area para un automata celular.", CelAre);
////////////////////////////////////////////////////////////////////////////
```

Set CelRul

```
////////////////////////////////////////////////////////////////////////////
Set CelRul = SetOfSet(
  RuleBuild([["OO => ··"]]),   // Muerte por superpoblacion

  RuleBuild([["O => ·",
              "O    ·"]]),     // Muerte por superpoblacion

  RuleBuild([["··· => ···",
              "·O·    ···",
```

```
              "···    ···"]]), // Muerte por soledad

  RuleBuild([["O·O => ·O·",
              "···    O·O"]]), // Reproduccion hacia abajo

  RuleBuild([["··· => O·O",
              "O·O    ·O·"]]), // Reproduccion hacia arriba

  RuleBuild([["O· => ·O",
              "··    O·",
              "O·    ·O"]]),   // Reproduccion lateral

  RuleBuild([["·O => O·",
              "··    ·O",
              "·O    O·"]]),   // Reproduccion lateral

  RuleBuild([["O· => ··",
              "··    ·O"]]),   // Movimiento diagonal

  RuleBuild([["·O => ··",
              "··    O·"]]),   // Movimiento diagonal

  RuleBuild([["·· => ·O",
              "O·    ··"]]),   // Movimiento diagonal

  RuleBuild([["·· => O·",
              "·O    ··"]]));  // Movimiento diagonal
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Base de reglas para probar el comportamiento de un automata celular.",
CelAre);
////////////////////////////////////////////////////////////////////////////
```

bat.tol de ChRules.RandRecursive

Base de reglas y area para una batalla entre bombarderos que lanzan bombas y lanzaderas para lanzar misiles. Aunque pareciendo diferentes las fuerzas y tecnicas de ambos contendientes son identicas y estan equilibradas. // Esta base de reglas representa una batalla fundamentalmente destructiva donde 2 bandos de bombarderos y lanzaderas de misiles se enfrentan de una forma equilibrada. // Esto es, todas las reglas a favor o en contra de uno de ellos tienen sus reglas equivalentes a favor o en contra del otro. // Adicionalmente, incluye pequeñas tactivas defensivas como: a) contrarrestar con una bomba un misil del contrario, b) contrarrestar con un misil una bomba del contrario, c) cerrar las defensas para evitar un impacto. // Tambien incluye ciertas complicaciones durante el combare, como por ejemplo: a) que bombas y misiles pueden sufrir desviaciones y b) que con al menos una defensa averiada ni los bombarderos y ni las lanzaderas pueden cerrarse ni moverse.

Declaraciones

Constantes

- `Set BatAre`

  Campo de batalla entre bombarderos y lanzaderas.

- `Set BatRul`

  Base de reglas de la batalla que incluyen reglas de ataque, de defensa, de efectos imprevistos como el desvio de misiles y bombas, etc.

Constantes

Set BatAre

```
////////////////////////////////////////////////////////////////////////////
Set BatAre = AreaBuild(
[["---------------",
  "(+)·(+)·(+)·(+)",
  "···············",
  "···············",
  "···············",
  "···············",
  "···············",
  "[^]·[^]·[^]·[^]",
  "---------------"]]);
////////////////////////////////////////////////////////////////////////////
PutDescription("Campo de batalla entre bombarderos y lanzaderas.", BatAre);
////////////////////////////////////////////////////////////////////////////
```

Set BatRul

```
////////////////////////////////////////////////////////////////////////////
Set BatRul = SetOfSet(
  RuleBuild([["O => ·",
              "·    O"]]),     // Bomba cae

  RuleBuild([["· => |",
              "|    ·"]]),     // Misil sube

  RuleBuild([["O => ·",
```

```
            "-    -"]]),     // Bomba se pierde sin causar daños
RuleBuild([["- => -",
            "|    ·"]]),     // Misil se pierde sin causar daños
RuleBuild([["o· => ··",
            "··    ·o"]]),   // Bomba se desvia a la derecha
RuleBuild([["·o => ··",
            "··    o·"]]),   // Bomba se desvia a la izquierda
RuleBuild([["·· => ·|",
            "|·    ··"]]),   // Misil se desvia a la derecha
RuleBuild([["·· => |·",
            "·|    ··"]]),   // Misil se desvia a la izquierda
RuleBuild([["o => ·",
            "|    ·"]]),     // Misil y bomba se destruyen mutuamente
RuleBuild([["o => ·",
            "[    ·"]]),     // Bomba destruye protector izquierdo
RuleBuild([["o => ·",
            "^    ·"]]),     // Bomba destruye lanzador de misiles
RuleBuild([["o => ·",
            "]    ·"]]),     // Bomba destruye protector derecho
RuleBuild([["( => ·",
            "|    ·"]]),     // Misil destruye protector izquierdo
RuleBuild([["+ => ·",
            "|    ·"]]),     // Misil destruye lanzador de bombas
RuleBuild([[") => ·",
            "|    ·"]]),     // Misil destruye protector derecho
RuleBuild([["(+) => ·()",
            "···    ···",
            "|··    |··"]]), // Borbardero intenta evitar misil izquierdo
RuleBuild([["(+) => ()·",
            "···    ···",
            "··|    ··|"]]), // Borbardero intenta evitar misil derecho
RuleBuild([["o·· => o··",
            "···    ···",
            "[^]    ·[]"]]), // Lanzadera intenta evitar bomba izquierda
RuleBuild([["··o => ··o",
            "···    ···",
            "[^]    []·"]]), // Lanzadera intenta evitar bomba derecha
RuleBuild([["(+) => (+)",
            "···    ·o·",
            "·|·    ·|·"]]), // Borbardero intenta evitar misil central
RuleBuild([["·o· => ·o·",
            "···    ·|·",
            "[^]    [^]"]]), // Lanzadera intenta evitar bomba central
RuleBuild([["()· => (+)"]]), // Bombardero se abre para bombardear
RuleBuild([["·() => (+)"]]), // Bombardero se abre para bombardear
RuleBuild([["[]· => [^]"]]), // Lanzadera se abre para disparar
RuleBuild([["·[] => [^]"]]), // Lanzadera se abre para disparar
RuleBuild([["+ => +",
            "·    o",
            "·    ·",
            "·    ·",
            "·    ·",
```

```
                ".    ."
                "^    ^"]]),     // Borbardero ataca lanzadera

    RuleBuild([["+ => +",
                ".    ."
                ".    ."
                ".    ."
                ".    ."
                ".    |"
                "^    ^"]]),     // Lanzadera ataca borbardero

    RuleBuild([["+ => +",
                ".    O"
                ".    ."
                ".    ."
                ".    ."
                ".    ."
                "(    ("]]),     // Borbardero ataca protector izquierdo

    RuleBuild([["+ => +",
                ".    O"
                ".    ."
                ".    ."
                ".    ."
                ".    ."
                ")    )"]]),     // Borbardero ataca protector derecho

    RuleBuild([["( => (",
                ".    ."
                ".    ."
                ".    ."
                ".    ."
                ".    |"
                "^    ^"]]),     // Lanzadera ataca protector izquierdo

    RuleBuild([[") => )",
                ".    ."
                ".    ."
                ".    ."
                ".    ."
                ".    |"
                "^    ^"]]));    // Lanzadera ataca protector derecho
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Base de reglas de la batalla que incluyen reglas de ataque, de defensa,
de efectos imprevistos como el desvio de misiles y bombas, etc.",
BatAre);
////////////////////////////////////////////////////////////////////////////
```

wal.tol de ChRules.RandRecursive

Base de reglas y area para derribar un muro con misiles. Es una base de reglas fundamentalmente destructiva, donde una serie de lanzaderas de misiles tratan de derribar un muro en el que a su vez sus elementos son explosivos y al recibir un impacto pueden producir cadenas de destruccion que se transmiten a sus elementos adyacentes en secuencia. // Esta base de reglas incluye un ultima regla de deteccion del final del proceso de deteccion de reglas y que construye la frase clasica de game over,

Declaraciones

Constantes

- `Set WalAre`
  Campo de batalla entre bombarderos y lanzaderas.
- `Set WalRul`
  Base de reglas para el derribo sistematico de un muro con 2 tipos de elementos grandes y pequeños hasta su derribo final.

Constantes

Set WalAre

```
////////////////////////////////////////////////////////////////////////////
Set WalAre = AreaBuild(
[["---------------",
  "·ooooooooooooo·",
  "··o·o·o·o·o·o··",
  "···············",
  "···············",
  "···············",
  "···············",
  "[^]·[^]·[^]·[^]",
  "---------------"]]);
////////////////////////////////////////////////////////////////////////////
PutDescription("Campo de batalla entre bombarderos y lanzaderas.", WalAre);
////////////////////////////////////////////////////////////////////////////
```

Set WalRul

```
////////////////////////////////////////////////////////////////////////////
Set WalRul = SetOfSet(
  RuleBuild([["· => |",
              "|    ·"]]),       // Misil sube

  RuleBuild([["··· => |||",
              "·|·    ···"]]),   // Misil se despliega en 3 cabezas

  RuleBuild([["- => -",
              "|    ·"]]),       // Misil se pierde sin causar daños

  RuleBuild([["o => 0",
              "|    ·"]]),       // Misil impacta e inicio de explosion

  RuleBuild([["0 => ·",
              "|    ·"]]),       // Misil impacta seco y destructivo
```

```
  RuleBuild([["o0 => 0·"]]),     // Explosion hacia la izquierda
  RuleBuild([["0o => ·0"]]),     // Explosion hacia la derecha
  RuleBuild([["o => 0",
              "0    ·"]]),       // Explosion hacia a arriba
  RuleBuild([["0 => ·",
              "o    0"]]),       // Explosion hacia a bajo
  RuleBuild([["0 => 0",
              ".     ·",
              ".     ·",
              ".     ·",
              ".     |",
              "^     ^"]]),     // Dispara corto sobre elemento grande
  RuleBuild([["o => o",
              ".     ·",
              ".     ·",
              ".     ·",
              ".     |",
              "^     ^"]]),      // Dispara corto sobre elemento pequeño

  RuleBuild([["0 => 0",
              ".     ·",
              ".     ·",
              ".     ·",
              ".     ·",
              ".     |",
              "^     ^"]]),     // Dispara largo sobre elemento grande
  RuleBuild([["o => o",
              ".     ·",
              ".     ·",
              ".     ·",
              ".     ·",
              ".     |",
              "^     ^"]]),      // Dispara largo sobre elemento pequeño
  RuleBuild([["--------------- => ---------------",
              "...............    G A M E O V E R"]])); // Fin del juego
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Base de reglas para el derribo sistematico de un muro con 2 tipos de
elementos grandes y pequeños hasta su derribo final.",
WalAre);
////////////////////////////////////////////////////////////////////////////
```

inc.tol de ChRules.RandRecursive

Inclusion de ficheros de funciones especificas de aplicacion

Declaraciones

Inclusiones de aplicación

- `Set rndInc`
  Reglas para probar comportamiento aleatorio.
- `Set cuaInc`
  Reglas de crecimiento cuadriaculando un area.
- `Set celInc`
  Reglas de un automata celular.
- `Set batInc`
  Reglas de batalla con misiles y bombas.
- `Set walInc`
  Reglas de para derribar un muro con misiles.

Set rndInc

```
////////////////////////////////////////////////////////////////////////////
Set  rndInc = Include("app/rnd.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas para probar comportamiento aleatorio.", rndInc);
////////////////////////////////////////////////////////////////////////////
```

Set cuaInc

```
////////////////////////////////////////////////////////////////////////////
Set  cuaInc = Include("app/cua.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas de crecimiento cuadriaculando un area.", cuaInc);
////////////////////////////////////////////////////////////////////////////
```

Set celInc

```
////////////////////////////////////////////////////////////////////////////
Set  celInc = Include("app/cel.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas de un automata celular.", celInc);
////////////////////////////////////////////////////////////////////////////
```

Set batInc

```
////////////////////////////////////////////////////////////////////////////
Set  batInc = Include("app/bat.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas de batalla con misiles y bombas.", barInc);
////////////////////////////////////////////////////////////////////////////
```

Set wallnc

```
////////////////////////////////////////////////////////////////////////////
Set  walInc = Include("app/wal.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas de para derribar un muro con misiles.", walInc);
////////////////////////////////////////////////////////////////////////////
```