# make.tol de 3d.SquaresPuzzle

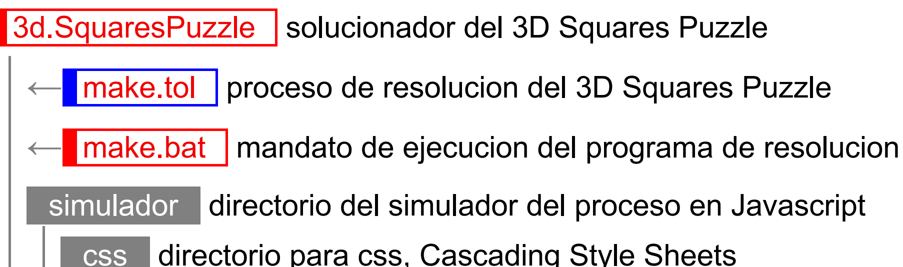
Solucionador del juego llamado 3D Squares Puzzle. Resuelve de forma recursiva un puzle de 9 piezas que, a pesar de su aparente sencillez, no es trivial. Encuentra 4 soluciones que, en el fondo, son la misma solucion pero con 1, 2 o 3 rotaciones de 90 grados. Las piezas de este puzle se simulan mediante tiras de 3x3 caracteres cada una y, en vez de codificar el tipo de insecto, se utilizan letras para el color fundamental del insecto, asi por ejemplo, V, verde mayuscula, para la cabeza del saltamontes verde y v, verde minuscula, para el cuerpo del saltamontes verde o A, amarillo mayuscula, para la cabeza del abejorro amarillo y a, amarillo minuscula, para el cuerpo del abejorro amarillo. Todo el codigo fuente de este solucionador esta desarrollado en un unico fichero que consta de varios grupos de funciones: a) funciones basicas, b) fuenciones de piezas, c) funciones de variaciones de piezas (giros) y d) funciones de busqueda de soluciones.

Este solucionador del 3D Squares Puzzle emplea una estrategia de busqueda que poda ramas en base a las restricciones de las piezas. Las restricciones son las siguientes: a) La primera pieza de la esquina superior izquierda no sufre restricciones. b) Para la seleccion segunda y tercera pieza de la linea superior hay que tener en cuenta la restriccion de conincidencia con la pieza mas a su izquierda. c) Para la seleccion segunda y tercera pieza de la columna izquierda hay que tener en cuenta la restriccion de conincidencia con su pieza de encima. d) Las otras 4 piezas no nombradas anteriormente sufren 2 restricciones, la de coincidencia con su pieza superior y la de coincidencia con su pieza mas a la izquierda.

Para entender el proceso de solucion del 3D Squares Puzzle que se emplea hay que tener en cuenta los siguientes conceptos: a) La resolucion del problema avanza de arriba hacia abajo y de izquierda a derecha. b) Para cada posicion se abren tantas ramas de exploracion como piezas, que todavia no se hayan puesto y que con una adecuada rotacion, cumplan las restricciones de coincidencia que le puedan imponer su posible pieza superior y la posible pieza a su izquierda. c) Si en una determinada rama de exploracion las restricciones impiden ya poner ninguna de las piezas que quedan, entonces la rama se poda. d) El problema queda resuelto cuando todas las piezas se han puesto, adecuadamente rotadas y cumpliendo sus restricciones de coincidencia.

La salida de este programa solucionador, si en la funciones SolCua() se quita el comentario a la funcion View() y adecuadamente transformada, permite la reproduccion del proceso de busqueda completo con Javascript y el conjunto de las piezas reales, ya no con mapas de caracteres.

# Árbol de ficheros



← puzle3d.css css para el simulador del puzle 3d

src directorio de codigo fuente Javascript

← simulator.js para simular visualmente la resolucion del puzle 3D

← simulatordb.js banco de pasos de resolucion del puzle 3D como un array

→ 3dsquarespuzzleproblema.png fotografia real del problema del puzle como punto de partida

→ 3dsquarespuzzlenumeracion.png imagen de la numeracion de las piezas del puzle 3D Square

→ 3dsquarespuzzlesolucion.png fotografia de la solucion con las piezas bien colocadas

→ 3dsquarespuzzle4soluciones.png las 4 posibles soluciones que son rotaciones del puzle 3D

→ 3dsquarespuzzlereal.png puzle 3D real con las piezas colocadas como indica Tol

→ simulator.html simulador del proceso de resolucion del 3D Squares Puzzle

→ startlog.txt fichero de log de un proceso completo de busqueda de solucion

→ 3d_squarespuzzle.pdf documento de las funciones del solucionador del 3D Puzzle

# Declaraciones

### Constantes

- `Set PieAll`
  Todas las piezas codificadas como caracteres.

### Funciones: Basicas

- `Real EvalWhile(Set set, Code fun)`
  Para todo elemento del set aplica la funcion fun pero, a diferencia de EvalSet(), solo retorna el cardinal del set. Por ello consume mucha menos memoria que EvalSet(), pero con mucha menos funcionalidad y es adecuado para procesos recursivos con muchas iteraciones donde no se necesita el conjunto de retorno. Notese que esta version de EvalWhile() espera que fun() sea una funcion que retorna un tipo Real, si bien los elementos del conjunto set pueden ser de cualquier tipo.

### Funciones: PIE(za)

- `Real PieVer(Text pie)`
  Visualiza una pieza por pantalla como su conjunto de caracteres.

- `Text PieRotDer(Text pie)`
  Rota a la derecha una pieza.

- `Text PieRot180(Text pie)`
  Rota 180º una pieza.

- `Text PieRotIzq(Text pie)`
  Rota a la izquierda una pieza.

- `Real PieCmbNor(Text uno, Text dos)`
  Si uno combina por el norte con dos.
- `Real PieCmbEst(Text uno, Text dos)`
  Si uno combina por el este con dos.
- `Real PieCmbSur(Text uno, Text dos)`
  Si uno combina por el sur con dos.
- `Real PieCmbOes(Text uno, Text dos)`
  Si uno combina por el oeste con dos.

### Funciones: VAR(iaciones)

- `Set VarAll(Text pie)`
  Retorna la pieza original y sus 3 giros.

### Funciones: SOL(uciones)

- `Real SolVer(Set sol)`
  Ver una solucion completa.
- `Real SolBue(Set sol)`
  Mira si una solucion es buena, que cumple con las restricciones.
- `Real SolCua(Set entSol, Set entPie)`
  Resuelve el 3D Squares Puzzle.

### Proceso

- `Real makSol`
  Ejecuta la resolucion del 3D Squares Puzzle.

### Pruebas

- `Real tstFun`
  Comprueba algunas de las funciones.

# Constantes

## Set PieAll

```
//////////////////////////////////////////////////////////////////////////////
Set  PieAll =
[[
   ".r."+ // Las minusculas son los cuerpos y las mayusculas las cabezas
   "M1v"+ // El numero del interior 1, 2,..., 9 es el numero de la pieza
   ".A.", // Los puntos en las esquinas no se emplean para nada

   ".a."+ // Los colores son Rr rojo Mm morado Aa amarillo Vv verde
   "A2v"+ // Los bichos son mariquita roja, araña morada, abejorro amarillo y
   ".R.", // saltamontes verde (aunque tenga partes naranja)

   ".R."+ // Por ejemplo:
   "V3m"+ // V verde mayuscula cabeza del saltamontes verde
   ".a.", // a amarillo minuscula cuerpo del abejorro amarillo

   ".R."+ // R rojo mayuscula cabeza de la mariquita roja
   "M4v"+ // v verde minuscula cuerpo del saltamontes verde
   ".A.", // A amarillo mayuscula cabeza del abejorro amarillo
```

```
    ".R."+
    "M5r"+ // M morado mayuscula cabeza de la araña morada
    ".v.",

    ".m."+ // m morado minuscula cuerpo de la araña morada
    "A6R"+
    ".v.",

    ".a."+
    "m7r"+
    ".v.",

    ".r."+ // r rojo minuscula cuerpo de la mariquita roja
    "v8M"+
    ".A.",

    ".v."+
    "M9m"+
    ".A."
]];
//////////////////////////////////////////////////////////////////////////////
PutDescription("Todas las piezas codificadas como caracteres.", PieAll);
//////////////////////////////////////////////////////////////////////////////
```

## Real EvalWhile()

```
//////////////////////////////////////////////////////////////////////////////
Real EvalWhile(Set set,
               Code fun)
//////////////////////////////////////////////////////////////////////////////
{
  Real num = 1;         // Contador de elementos
  Real max = Card(set); // Maximo del set

  Real While(LE(num, max), // Hasta fin del set
  {
    Anything ele = set[num];
    Real     res = fun(ele);
    Real    (num:= Copy(num) + 1);

    TRUE
  });
  max
};
//////////////////////////////////////////////////////////////////////////////
PutDescription(
"Para todo elemento del set aplica la funcion fun pero, a diferencia de
EvalSet(), solo retorna el cardinal del set.
Por ello consume mucha menos memoria que EvalSet(),
pero con mucha menos funcionalidad y es adecuado para procesos recursivos
con muchas iteraciones donde no se necesita el conjunto de retorno.
Notese que esta version de EvalWhile() espera que fun() sea una funcion que
retorna un tipo Real, si bien los elementos del conjunto set pueden ser
de cualquier tipo.",
EvalWhile);
//////////////////////////////////////////////////////////////////////////////
```

## Real PieVer()

```
//////////////////////////////////////////////////////////////////////////////
Real PieVer(Text pie)
//////////////////////////////////////////////////////////////////////////////
{
  Text WriteLn(Sub(pie, 1, 3)+"\n"+
               Sub(pie, 4, 6)+"\n"+
               Sub(pie, 7, 9)+"\n"+
               "---");
```

```
  TRUE
};
//////////////////////////////////////////////////////////////////////////////
PutDescription(
"Visualiza una pieza por pantalla como su conjunto de caracteres.",
PieVer);
//////////////////////////////////////////////////////////////////////////////
```

## Text PieRotDer()

```
//////////////////////////////////////////////////////////////////////////////
Text PieRotDer(Text pie)
//////////////////////////////////////////////////////////////////////////////
{
  Text nor = Sub(pie, 2, 2);
  Text oes = Sub(pie, 4, 4);
  Text est = Sub(pie, 6, 6);
  Text sur = Sub(pie, 8, 8);

  Text cen = Sub(pie, 5, 5); // Conserva el numero de pieza

  Text rot = "."+oes+"."+
             sur+cen+nor+
             "."+est+".";
  rot
};
//////////////////////////////////////////////////////////////////////////////
PutDescription(
"Rota a la derecha una pieza.",
PieRotDer);
//////////////////////////////////////////////////////////////////////////////
```

## Text PieRot180()

```
//////////////////////////////////////////////////////////////////////////////
Text PieRot180(Text pie) //
//////////////////////////////////////////////////////////////////////////////
{ PieRotDer(PieRotDer(pie)) };
//////////////////////////////////////////////////////////////////////////////
PutDescription(
"Rota 180º una pieza.",
PieRot180);
//////////////////////////////////////////////////////////////////////////////
```

## Text PieRotIzq()

```
//////////////////////////////////////////////////////////////////////////////
Text PieRotIzq(Text pie)
//////////////////////////////////////////////////////////////////////////////
{ PieRotDer(PieRotDer(PieRotDer(pie))) };
//////////////////////////////////////////////////////////////////////////////
PutDescription(
"Rota a la izquierda una pieza.",
PieRotIzq);
//////////////////////////////////////////////////////////////////////////////
```

## Real PieCmbNor()

```
//////////////////////////////////////////////////////////////////////////////
Real PieCmbNor(Text uno,
               Text dos)
//////////////////////////////////////////////////////////////////////////////
```

```
{
  Text nor = Sub(uno, 2, 2);
  Text sur = Sub(dos, 8, 8);
  EQ(32, Abs(ASCII(nor)-ASCII(sur))) // (a-A) = (m-M) = (r-R) = (v-V) = 32
};
//////////////////////////////////////////////////////////////////////////////
PutDescription(
"Si uno combina por el norte con dos.",
PieCmbNor);
//////////////////////////////////////////////////////////////////////////////
```

## Real PieCmbEst()

```
//////////////////////////////////////////////////////////////////////////////
Real PieCmbEst(Text uno,
               Text dos)
//////////////////////////////////////////////////////////////////////////////
{
  Text est = Sub(uno, 6, 6);
  Text oes = Sub(dos, 4, 4);
  EQ(32, Abs(ASCII(est)-ASCII(oes))) // (a-A) = (m-M) = (r-R) = (v-V) = 32
};
//////////////////////////////////////////////////////////////////////////////
PutDescription(
"Si uno combina por el este con dos.",
PieCmbEst);
//////////////////////////////////////////////////////////////////////////////
```

## Real PieCmbSur()

```
//////////////////////////////////////////////////////////////////////////////
Real PieCmbSur(Text uno,
               Text dos)
//////////////////////////////////////////////////////////////////////////////
{
  Text sur = Sub(uno, 8, 8);
  Text nor = Sub(dos, 2, 2);
  EQ(32, Abs(ASCII(sur)-ASCII(nor))) // (a-A) = (m-M) = (r-R) = (v-V) = 32
};
//////////////////////////////////////////////////////////////////////////////
PutDescription(
"Si uno combina por el sur con dos.",
PieCmbSur);
//////////////////////////////////////////////////////////////////////////////
```

## Real PieCmbOes()

```
//////////////////////////////////////////////////////////////////////////////
Real PieCmbOes(Text uno,
               Text dos)
//////////////////////////////////////////////////////////////////////////////
{
  Text oes = Sub(uno, 4, 4);
  Text est = Sub(dos, 6, 6);
  EQ(32, Abs(ASCII(oes)-ASCII(est))) // (a-A) = (m-M) = (r-R) = (v-V) = 32
};
//////////////////////////////////////////////////////////////////////////////
PutDescription(
"Si uno combina por el oeste con dos.",
PieCmbOes);
//////////////////////////////////////////////////////////////////////////////
```

## Set VarAll()

```
////////////////////////////////////////////////////////////////////////////////
Set  VarAll(Text pie)
////////////////////////////////////////////////////////////////////////////////
{
  SetOfText(pie, PieRotDer(pie), PieRot180(pie), PieRotIzq(pie))
};
////////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna la pieza original y sus 3 giros.",
VarAll);
////////////////////////////////////////////////////////////////////////////////
```

## Real SolVer()

```
////////////////////////////////////////////////////////////////////////////////
Real SolVer(Set sol)
////////////////////////////////////////////////////////////////////////////////
{
  Text Write("\n\nSOLUCION:\n\n"+
      "   "+"---"              +"  "+"---"              +"  "+"---"
  Set cic = For(0, 2, Real(Real lin)
  {
    Real li3 = lin * 3;
    Text Write(
      "  |"+Sub(sol[li3+1],1,3)+"|"+Sub(sol[li3+2],1,3)+"|"+Sub(sol[li3+3],1,3)
      "  |"+Sub(sol[li3+1],4,6)+"|"+Sub(sol[li3+2],4,6)+"|"+Sub(sol[li3+3],4,6)
      "  |"+Sub(sol[li3+1],7,9)+"|"+Sub(sol[li3+2],7,9)+"|"+Sub(sol[li3+3],7,9)
      "   "+"---"              +"  "+"---"              +"  "+"---"
    li3
  });
  Text WriteLn("");
  Card(cic)
};
////////////////////////////////////////////////////////////////////////////////
PutDescription(
"Ver una solucion completa.",
SolVer);
////////////////////////////////////////////////////////////////////////////////
```

## Real SolBue()

```
////////////////////////////////////////////////////////////////////////////////
Real SolBue(Set sol)
////////////////////////////////////////////////////////////////////////////////
{
  Real crd = Card(sol);

  If(   LE(crd, 1),      TRUE,      // Con 0 o 1 piezas siempre esta bien
  If(   LE(crd, 3),      PieCmbEst(sol[crd-1], sol[crd]),   // El 2 y el 3
  If(Or(EQ(crd, 4),
        EQ(crd, 7)),     PieCmbSur(sol[crd-3], sol[crd]),   // El 4 y 7
                     And(PieCmbSur(sol[crd-3], sol[crd]),   // El 5, 6, 8 y 9
                         PieCmbEst(sol[crd-1], sol[crd])))))
};
////////////////////////////////////////////////////////////////////////////////
PutDescription(
"Mira si una solucion es buena, que cumple con las restricciones.",
SolBue);
////////////////////////////////////////////////////////////////////////////////
```

## Real SolCua()

```
////////////////////////////////////////////////////////////////////////////////
Real SolCua(Set entSol,
            Set entPie)
////////////////////////////////////////////////////////////////////////////////
{
  Real crdSol = Card(entSol);
  Text Write(FormatReal(crdSol,"%.0lf")); // Visualiza el nº de piezas puestas
  Set  View([[entSol]], "");              // Visualiza las piezas puestas

  If(EQ(crdSol, 9), SolVer(entSol), // Ha encontrado una solucion
  {                 // Busca soluciones
    EvalWhile(entPie, Real(Text unoPie)
    {
      Set salPie = entPie - [[unoPie]];
      Set varPie = VarAll(unoPie);
      EvalWhile(varPie, Real(Text unoVar)
      {
        Set salSol = entSol << [[ unoVar ]];
        If(SolBue(salSol), SolCua(salSol, salPie), FALSE)
      })
    })
  })
};
////////////////////////////////////////////////////////////////////////////////
PutDescription(
"Resuelve el 3D Squares Puzzle.",
SolCua);
////////////////////////////////////////////////////////////////////////////////
```

## Real makSol

```
////////////////////////////////////////////////////////////////////////////////
Real makSol = SolCua(Empty, PieAll);
////////////////////////////////////////////////////////////////////////////////
PutDescription("Ejecuta la resolucion del 3D Squares Puzzle.", makSol);
////////////////////////////////////////////////////////////////////////////////
```

## Real tstFun

```
////////////////////////////////////////////////////////////////////////////////
Real tstFun = If(TRUE, FALSE, // Poner a false para ejecutar las pruebas
{
  Text WriteLn(PieAll[1]);

  Text PieVer(PieAll[1]);
  Text PieVer(PieRotDer(PieAll[1]));
  Text PieVer(PieRot180(PieAll[1]));
  Text PieVer(PieRotIzq(PieAll[1]));
  Real PieCmbOes(PieAll[1], PieAll[3])
});
////////////////////////////////////////////////////////////////////////////////
PutDescription("Comprueba algunas de las funciones.", tstFun);
////////////////////////////////////////////////////////////////////////////////
```