
## ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

## GRADO EN INGENIERÍA INFORMÁTICA – TECNOLOGÍAS INFORMÁTICAS

# TANGRAM RECTÁNGULOS – PROGRAMACIÓN DECLARATIVA

**Realizado por FRANCISCO JESÚS BELMONTE PINTRE**

**Dirigido por CARMEN GRACIANI DÍAZ**

**Departamento de CIENCIAS DE LA COMPUTACIÓN Y DE LA INTELIGENCIA ARTIFICIAL**

**Sevilla, 2017-18**

# Contenido

# 1: FUNCIONAMIENTO DEL JUEGO

**Tecla 1:** Seleccionar la pieza verde.

**Tecla 2:** Seleccionar la pieza azul.

**Tecla 3:** Seleccionar la pieza roja.

**Tecla 4:** Seleccionar la pieza naranja.

**Tecla 5:** Seleccionar la pieza rosa.

**Tecla 6:** Seleccionar la pieza amarilla.

Con **Clic Izquierdo**, teniendo una de las 6 piezas seleccionadas, podemos ver una ilustración de cómo quedaría la pieza si la colocásemos en el asterisco en el que pinchemos.

Con **Ctrl**, podemos dejar colocada, de manera temporal y en el lugar en el que hemos pinchado con el **Click Izquierdo** la pieza que teníamos seleccionado.

La pieza colocada de manera temporal con **Ctrl**, puede ser movida de sitio en cualquier momento volviendo a pulsar sobre la tecla de las 6 que corresponda y eligiendo otro de los puntos disponibles para colocarla usando **Clic Izquierdo**

También es posible volver a sacarla del tablero, para ganar espacio en el caso en que queramos buscar otra estrategia, teniéndola seleccionada y pulsando **Escape**.

Para deseleccionar una pieza que actualmente está seleccionada, pulsamos **Escape.**

Pulsando **Enter**, en vez de **Ctrl**, lo que haremos es colocar definitivamente la pieza, y por tanto no la podremos volver a mover más.

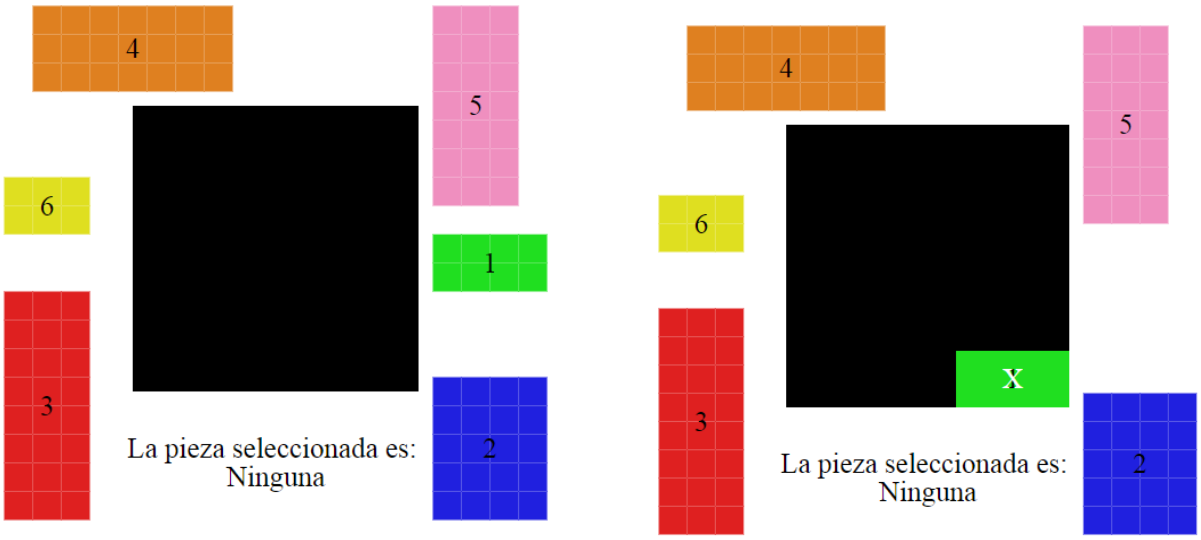
Por tanto, la principal diferencia entre **Ctrl** y **Enter** es que con **Ctrl** obtenemos una ilustración de la pieza en el lugar indicado de manera temporal, mientras que con **Enter** es definitivo y no se puede volver a recuperar la pieza, lo que se traduce en el movimiento definitivo de dicha pieza.

Las piezas se mostrarán inicialmente, en los alrededores del tablero para que el jugador pueda visualizar en todo momento que forma tiene la pieza. Cada una tendrá

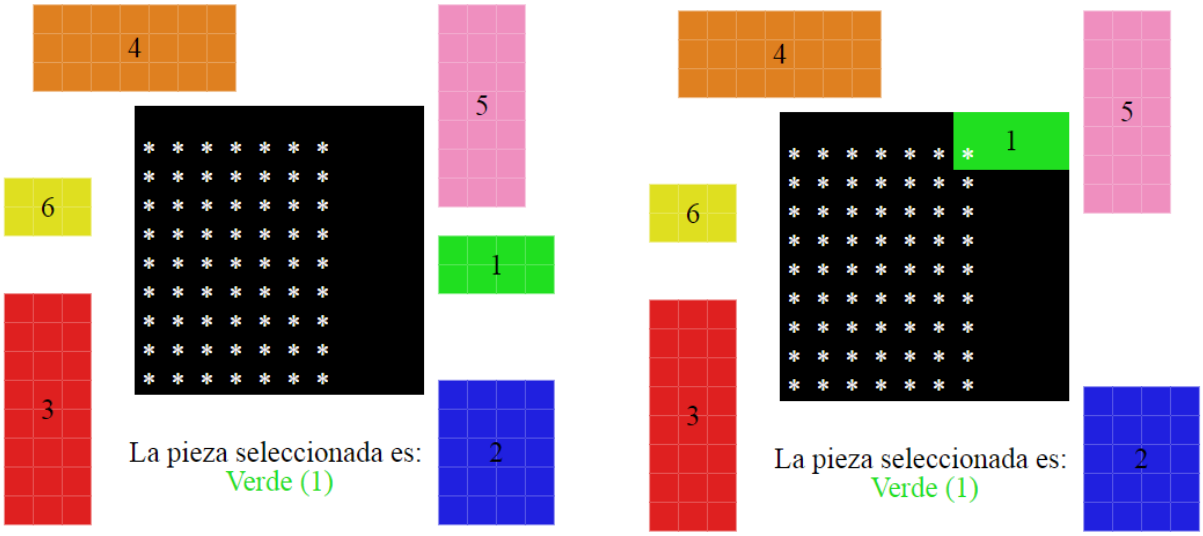
su enumeración en su centro (1, 2, 3, 4, 5, 6) para que el jugador sepa que tecla ha de pulsar para seleccionar cada una.

En la parte inferior se mostrara un texto donde se indicará la pieza que hay actualmente seleccionada, para que el jugador sepa en todo momento con que pieza está jugando en un instante determinado.

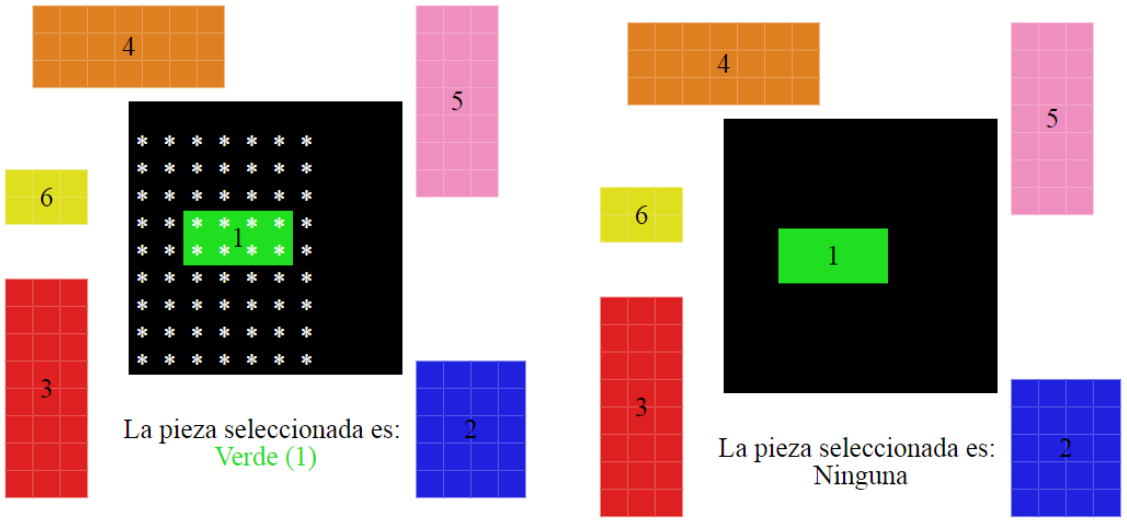
Cada vez que se coloque una pieza en el tablero, ya sea de manera temporal o definitiva, la misma pieza que había en el alrededor “a modo de ilustración” desaparece para generarse en la zona donde el jugador ha indicado que quería colocarla. Veamos un ejemplo:



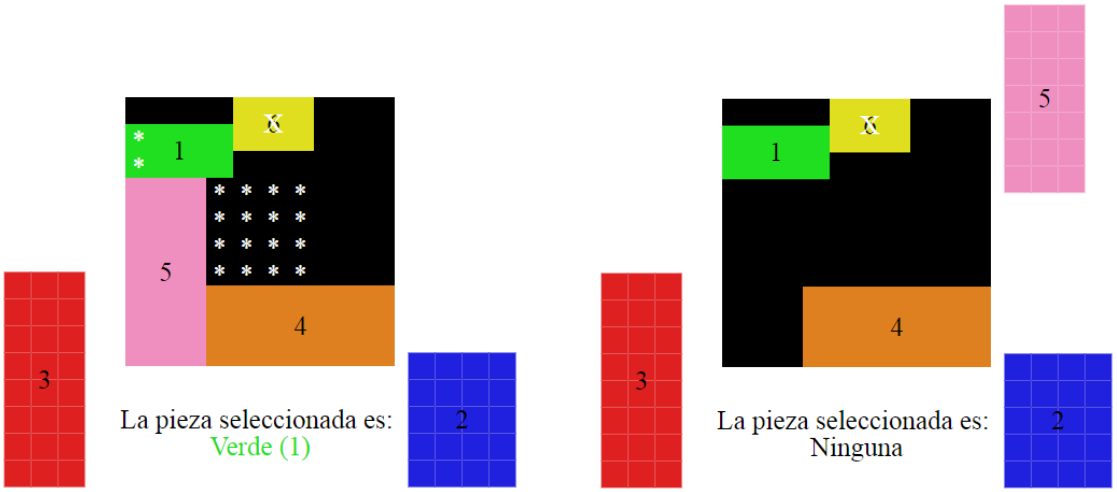

En la imagen de la izquierda vemos una inicialización del juego, sin haber tecleado absolutamente nada. A diferencia de la imagen de la derecha, donde la pieza verde ha sido colocada de manera definitiva (véase la X blanca sobre la enumeración, que indica una colocación definitiva) y la pieza seleccionada vuelve a “Ninguna”.




En la imagen de la izquierda podemos observar como la pieza 1 está seleccionada y los asteriscos funcionan a modo de esquinas inferiores izquierdas de la pieza en cuestión en los posibles sitios donde podría ser colocada sin que llegue a salirse del cuadrilátero negro y sin que llegue a pisar a otra pieza. En la imagen derecha vemos una ilustración de cómo quedaría la pieza 1 si la colocásemos en la zona en la que se ve en la imagen, igualmente se puede cambiar de sitio haciendo **Click Izquierdo** en cualquier otro asterisco.




En la imagen izquierda lo que hemos hecho es pinchar en otro asterisco distinto, y nos ha aparecido una ilustración de la pieza verde en una zona distinta del cuadrilátero. La imagen derecha muestra la pieza verde colocada de manera temporal y habiendo hecho uso del **Ctrl** para ello. Si se hubiese colocado de manera definitiva, tendría una X blanca encima.




En cualquier momento podemos mover una pieza de sitio, siempre y cuando no se haya colocado de manera definitiva, volviendo a pulsar su tecla numérica asociada y nos volverán a aparecer las posibilidades. En la imagen de la izquierda, estamos cambiando de sitio la pieza verde. La pieza 4 y 5 están colocados en sus respectivos sitios de manera temporal pudiéndose cambiar de sitio. La pieza 6 que posee una X blanca que indica que está colocada de manera definitiva y no puede moverse más de sitio. En la imagen derecha hemos sacado la pieza 5 (Rosa) del tablero pulsando escape cuando la teníamos seleccionada, y por tanto ha pasado de estar dentro del tablero a estar en los alrededores del tablero. Igualmente sigue siendo seleccionable y colocable.

Una victoria podría ser:

## 2: ESPECIFICACIONES DEL TANGRAM

Antes de entrar en materia, expliquemos un poco los type's y los data's utilizados:

```
data Pieza = Ninguna | Verde | Azul | Roja | Naranja | Rosa |
Amarilla deriving (Show,Eq)

type Casilla = (Double, Double)
type Mundo = ([Pieza], Pieza, [Casilla], [Casilla], [Casilla],
[Casilla], [Casilla], [Casilla], [Casilla])

piezas :: [Pieza]
piezas = [Verde,Azul,Roja,Naranja,Rosa,Amarilla]
```

Por un lado tenemos el data Pieza con todos los valores que puede tomar Pieza + "Ninguna". El caso de "Ninguna" se da durante la inicialización del juego, cuando no tenemos ninguna pieza seleccionada, además de cuando colocamos de manera temporal o definitiva una pieza en una zona, Pieza pasa a valer "Ninguna".

El caso de Casilla es un type de 2 coordenadas de tipo Double para definir puntos en un plano.

El type Mundo es el estado, el cual contiene, la lista de piezas disponible que se inicializa con todas, es decir, pasándole por parámetro de entrada "piezas" que contiene en una lista a todas. Lo siguiente que contiene es la pieza seleccionada/elegida, que en caso de que se acabe de inicializar el juego vale "Ninguna", y en el caso de colocar una pieza de manera temporal o definitiva también vale "Ninguna".

El tercero es la lista de casillas libres del tablero, es decir que están a negras. Se inicializa con la totalidad de las piezas del tablero, puesto que cuando ejecutamos el juego, el tablero está totalmente vacío y por ende todas las casillas son proclives a que se pueda intentar colocar una pieza sobre ellas. Por esta razón se inicializa con todas las casillas de tablero.

Las 6 restantes listas, una para cada pieza, contendrán los puntos en el plano donde está colocada cada pieza. En el caso de vacía, significa que la pieza aún no se ha colocado en el tablero.

La idea en general es construir un tablero de color negro de dimensiones 10x10, y el estado de esos 100 puntos en el tablero (si están libres u ocupados) se conocerá en todo momento puesto que dispondremos inicialmente de una lista con esos 100 puntos del tablero, y cada vez que se coloque una pieza en el tablero, esa lista se verá alterada

eliminando los puntos de esa lista que ahora forman parte de la pieza que se haya colocado. Esta lista formará parte del estado del problema.

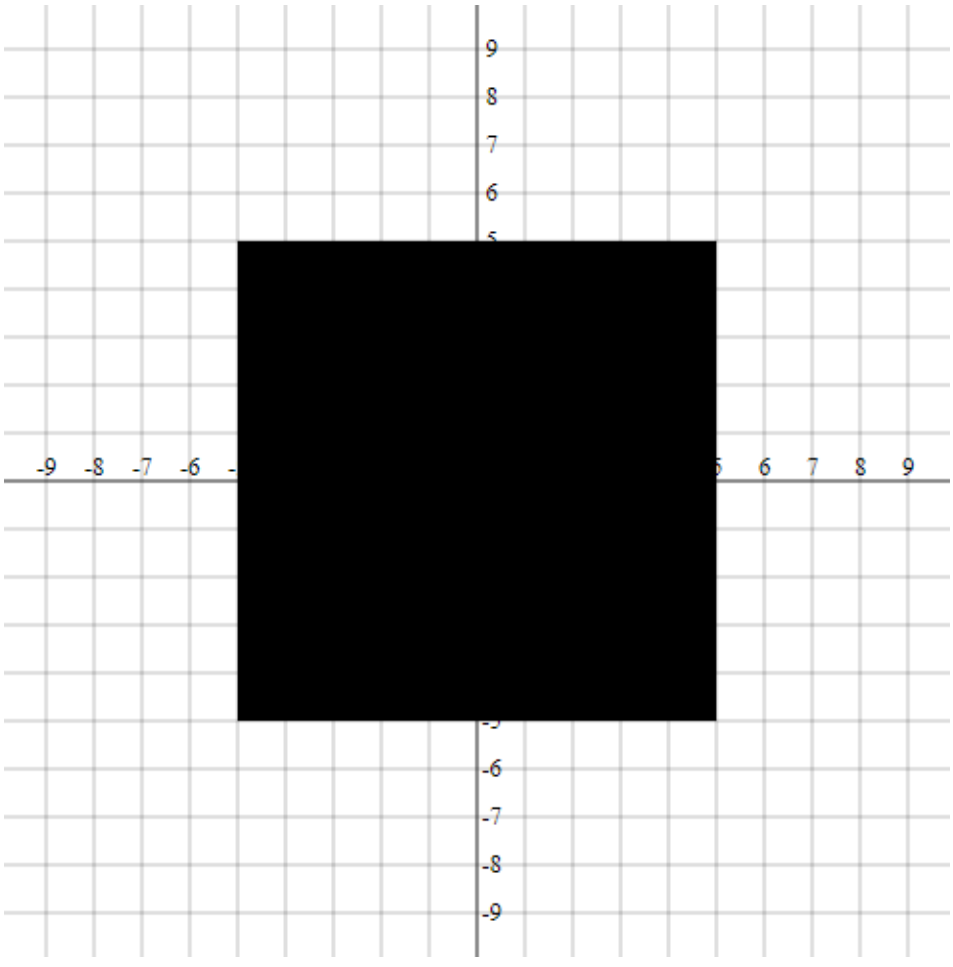
El tablero generado inicialmente, es el que se pasa a la lista de libres cuando inicializamos el estado, puesto que inicialmente el tablero está vacío y no hay ninguna pieza colocada, eso significa que los puntos del tablero y los puntos de la lista de libres han de ser los mismos.

Para generar el tablero y pintarlo, se ha utilizado la siguiente estructura que genera puntos en el plano de izquierda a derecha y de abajo a arriba, empezando en (-4.5,-4.5), que suma de 0 a 9 unidades en horizontal y luego aumenta en 1 en vertical para volver a repetir el proceso.

```
tablero :: [Casilla]
tablero = [(((-5.0)+1/2)+x,((-5.0)+1/2)+y)|x<-[0.0..9.0], y<-
[0.0..9.0]]

pintaTablero :: [Casilla] -> Picture
pintaTablero [] = blank
pintaTablero ((x,y):xs) = (translated x y (coloured black
(solidRectangle 1 1))) & (pintaTablero xs)

main :: IO()
main = drawingOf(pintaTablero tablero & coordinatePlane)
```



Acto seguido se han elaborado un total de 6 piezas con las siguientes dimensiones:

**Verde:** 4x2

**Azul:** 4x5

**Roja:** 3x8

**Naranja:** 7x3

**Rosa:** 3x7

**Amarilla:** 3x2

De tal forma que colocadas debidamente, el tablero de 10x10 queda rellenado completamente.

Para definir la dimensión de cada pieza se ha usado la siguiente estructura, la cual funciona parecido a la generadora del tablero, pues esta función se utilizará para construir los puntos de la pieza a partir de una esquina inferior izquierda dada, y a ella se le sumarán los puntos generados por los bucles anidados de esta función

```
dimensionPieza :: Pieza -> [Casilla]
dimensionPieza Verde = [(x, y)|x<-[0.0..3.0], y<-[0.0..1.0]]
dimensionPieza Azul = [(x, y)|x<-[0.0..3.0], y<-[0.0..4.0]]
dimensionPieza Roja = [(x, y)|x<-[0.0..2.0], y<-[0.0..7.0]]
dimensionPieza Naranja = [(x, y)|x<-[0.0..6.0], y<-[0.0..2.0]]
dimensionPieza Rosa = [(x, y)|x<-[0.0..2.0], y<-[0.0..6.0]]
dimensionPieza Amarilla = [(x, y)|x<-[0.0..2.0], y<-[0.0..1.0]]
```

Finalmente, para construir una pieza se ha utilizado la siguiente estructura, a la cual se le pasa por parámetro de entrada un punto que funcionará a modo de esquina inferior izquierda y a dicho punto se le van sumando los puntos generados por los 2 bucles anidados de la función dimensionPieza que son pasados a la función por segundo parámetro en forma de lista de Casilla, la cual es una tupla de 2 Double's

```
construirPieza :: Casilla -> [Casilla] -> [Casilla]
construirPieza (x, y) dimension = [(x+k, y+z)|(k, z)<-dimension]
```

## 3: ¿CÓMO SABER SI UNA DETERMINADA PIEZA SE PUEDE COLOCAR EN UN DETERMINADO LUGAR?

Como hemos mencionado anteriormente, se sabe en todo momento los puntos del cuadrilátero que son negros, o lo que es lo mismo, que están libres ya que disponemos en el estado de una lista que contiene aquellos puntos libres.

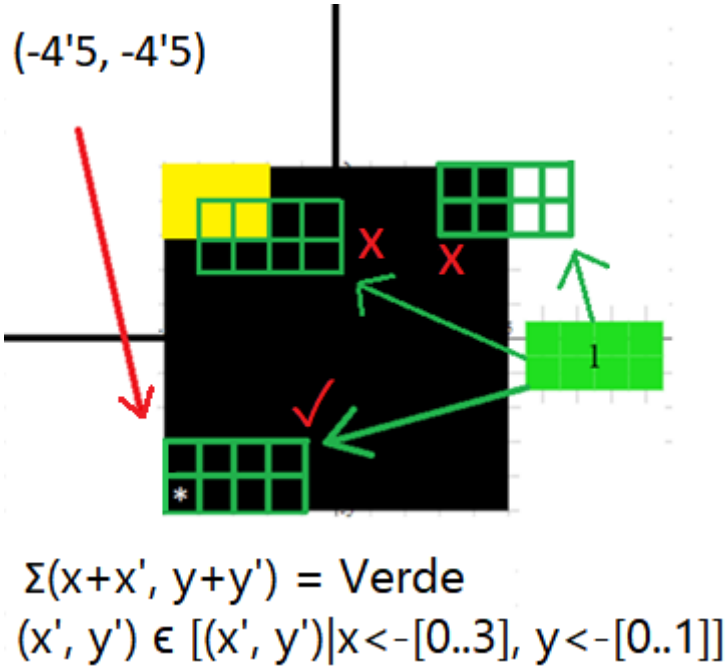
Por tanto, la idea consiste en recorrer la lista de puntos y por cada uno de ellos, construir la pieza que queramos saber si es posible colocarla o no, utilizando dicho punto como esquina inferior izquierda de la pieza y generando el resto de puntos de la pieza que se obtiene al sumarle los puntos generados por la función ***"dimensionPieza"***.

Por poner un ejemplo con la pieza de color verde de dimensiones (4x2), si el tablero está actualmente vacío, el punto que representa la esquina inferior izquierda del tablero está vacío y por tanto dentro de la lista de vacíos. Utilizando dicho punto como esquina inferior izquierda de la pieza verde que quiero saber si puedo colocar ahí, y moviéndome 3 posiciones más hacia la derecha, sigue estando vacío, pero además si hago exactamente lo mismo justo encima (1 unidad más en el eje Y) y me fijo en las 4 posiciones, también están vacías e incluidas dentro de la lista de vacías, ergo ésta es una de las zonas donde sí se puede colocar la pieza verde.

Para moverme 3 posiciones a la derecha desde la esquina inferior izquierda de la posible pieza colocada, y hacer lo mismo justo en el nivel superior, se utiliza un doble bucle anidado del tipo: [(x', y')|x'<-[0.0..3.0], y'<-[0.0..1.0]], que proporciona una serie de puntos que han de sumarse al punto que representa la esquina inferior izquierda de la pieza, y dicha suma la llevará a cabo la función ***"construirPieza"*** comentada antes.

De tal forma que podemos construir los posibles puntos que van a ser ocupados por la pieza, sumando a la coordenada x de la pieza la x' y a la coordenada y la y'. Esos 8 puntos que se generan, habrá que iterarlos y comprobar que efectivamente no se encuentran dentro de la lista de puntos libres que mencionamos anteriormente, y si esto se verifica significa que la pieza se puede colocar.

Veamos una ilustración:




Dispondremos de un método denominado ***"calculaOpciones"*** que recibe la lista actual de casillas vacías/libres/negras y la pieza de deseamos colocar, y lo que hace es iterar sobre cada una de las piezas de la lista utilizándolas a modo de esquina inferior izquierda de la pieza que deseo colocar, y comprobando el hecho de que si se construye la pieza, la totalidad de los puntos que la ocupan sigan perteneciendo a la lista de vacías

```
calculaOpciones :: [Casilla] -> Pieza -> [Casilla]
calculaOpciones _ Ninguna = []
calculaOpciones recuadrosLibres piezaElegida =
   [puntoPlano | puntoPlano <- recuadrosLibres, (calculaOpcionesAux
puntoPlano (dimensionPieza piezaElegida) recuadrosLibres)]

calculaOpcionesAux :: Casilla -> [Casilla] -> [Casilla] -> Bool
calculaOpcionesAux (x, y) dimension recuadrosLibres = and[(x+k, y+z)
`elem` recuadrosLibres| (k, z) <- dimension]
```

***"calculaOpciones"*** se apoya en ***"calculaOpcionesAux"*** que lo que hace es comprobar que todos y cada uno de los puntos en el plano que se generan por el hecho de construir la pieza, siguen perteneciendo a la lista de vacías.

## 4: QUITAR, AÑADIR, COLOREAR PIEZAS Y ACTUALIZAR LAS LISTAS CORRESPONDIENTES

Si la pieza está colocada, significa que se han descontado de la lista de los 100 puntos aquellos puntos que ahora forman parte de la pieza construida y colocada en el tablero, para el caso de la verde: 100 - 8 = 96

Para actualizar la lista de vacíos, utilizamos la siguiente estructura, en donde el método ***"quitaRecuadrosLibres"*** recibe la lista de libres/vacíos y la lista de casillas en donde se haya colocada la pieza y devuelve una nueva lista de Casilla habiendo quitado de las libres aquellas que están en el segundo parámetro.

```
quitaRecuadrosLibres :: [Casilla] -> [Casilla] -> [Casilla]
quitaRecuadrosLibres recuadrosLibres [] = recuadrosLibres
quitaRecuadrosLibres recuadrosLibres (x:xs) =
   (quitaRecuadrosLibres (quitaRecuadrosLibresAux recuadrosLibres x)
xs)


quitaRecuadrosLibresAux :: [Casilla] -> Casilla -> [Casilla]
quitaRecuadrosLibresAux [x] recuadroLibre = []
quitaRecuadrosLibresAux (x:xs) recuadroLibre
   |x == recuadroLibre = xs
   |otherwise = x:(quitaRecuadrosLibresAux xs recuadroLibre)
```

Si la pieza se coloca de manera temporal en el tablero, lo único que hay que hacer es llamar al método "quitaRecuadrosLibres" y actualizar las listas de vacíos, pero en el caso en que queramos colocar la pieza de manera definitiva, habría que, además de actualizar la lista de vacíos, eliminar la pieza que hemos colocado de manera definitiva de la lista de piezas disponible, que recordemos que era el primer parámetro del tipo Mundo.

Para ello tenemos la siguiente estructura, que recibe por entrada la lista de piezas disponible y la pieza que queremos quitar, y la elimina.

```
quitaPieza :: [Pieza] -> Pieza -> [Pieza]
quitaPieza [x] pieza = []
quitaPieza (x:xs) pieza
   |pieza == x = xs
   |otherwise = x:(quitaPieza xs pieza)
```

Por otra parte, se mencionó en el funcionamiento del juego que el jugador podía mover las piezas de un sitio a otro e incluso retirarlas del tablero para ganar visión, en

el caso en que no estuviesen colocadas de manera definitiva con **Enter**. Esto implica que de alguna manera, si el jugador decide retirar una ficha del tablero, esa lista de puntos que conforman la pieza ha de vaciarse, y si el jugador decide mover la pieza de una zona a otra, los valores de la lista cambiarán a otros valores, puesto que los puntos no serán los mismos si la pieza verde está colocada en la esquina inferior izquierda que en la esquina superior derecha o en el centro del tablero.

Para solucionar este problema, se ha utilizado una lista de puntos para cada una de las 6 piezas, que formarán parte del estado junto con la anterior lista que representaba los puntos libres del tablero. De tal forma que si la lista asociada a la pieza roja es una lista vacía, significa que no está colocada la pieza roja aún, y si no está vacía significa que sí está colocada y que los puntos en el tablero que ocupa dicha pieza son los contenidos en su lista asociada.

Ahora sólo faltaría llamar al método ***"pintaPieza"*** que por cada uno de esos puntos contenidos en la lista, coloque un solidRectangle de tamaño 1 x 1 en las coordenadas (x, y) que representan los puntos [(x,y),…] de su lista asociada y darle a cada uno de esos solidRectangle's el color asociado a la pieza, en el caso de la pieza Roja, se le dará el color rojo y para cada valor que puede tomar Pieza, habrá un método que me devolverá el color asociado, ejemplo: Pieza Roja -> Color rojo.

```
colorPieza :: Pieza -> Color
colorPieza Verde = green
colorPieza Azul = blue
colorPieza Roja = red
colorPieza Naranja = orange
colorPieza Rosa = pink
colorPieza Amarilla = yellow
```

En cuanto al método ***"pintaPieza"***, dispone de 3 parámetros de entrada. El primero de ellos es uno de los valores del data Pieza (Ninguna | Verde | Azul…) que nos servirá para elegir el color de los cuadrados que vamos a pintar, porque sólo con los puntos no se puede saber de qué color es el cuadrado que vamos a pintar en dicho punto. El segundo parámetro son la lista de puntos donde está colocada la pieza y, por tanto, hay que pintar los cuadrados de color. El último parámetro es la lista de piezas disponible que nos servirá para saber si tenemos que marcar la pieza con una X blanca (colocación de una pieza definitiva) o no, y es que una pieza en concreto estará definitivamente colocada, y por tanto marcada con una X, sí y sólo sí, ésta no se haya en la lista de piezas disponibles (1º parámetro del estado).

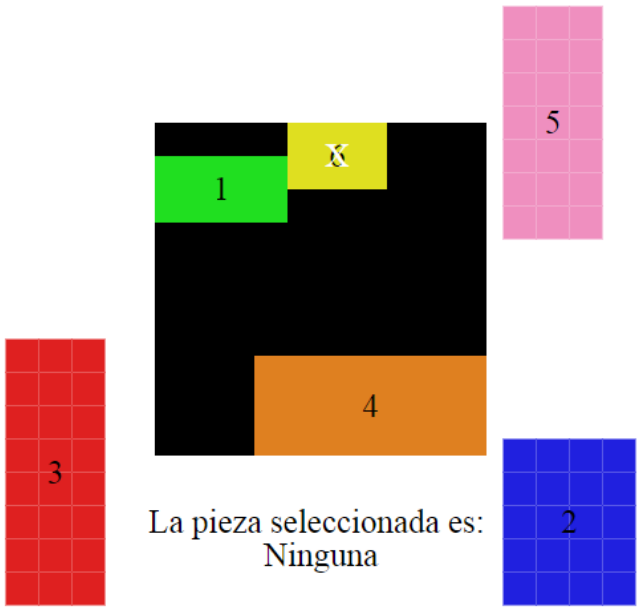
```
pintaPieza :: Pieza -> [Casilla] -> [Pieza] -> Picture
pintaPieza _ [] piezasRestantes = blank
pintaPieza pieza recuadrosPieza piezasRestantes=
   (ajustaEnumeracionPiezaColocada pieza (recuadrosPieza!!0) piezasRestantes) &
   pictures[translated x y (coloured (colorPieza pieza) (solidRectangle 1 1))|(x, y)<-
recuadrosPieza]

ajustaEnumeracionPiezaColocada :: Pieza -> Casilla -> [Pieza] -> Picture
ajustaEnumeracionPiezaColocada Verde (x, y) piezasRestantes
   |Verde `elem` piezasRestantes = (translated (x+1.5) (y+0.5) (coloured black (text "1")))
   |otherwise = (translated (x+1.5) (y+0.5) (coloured white (text "X"))) & (translated (x+1.5)
(y+0.5) (coloured black (text "1")))
ajustaEnumeracionPiezaColocada Azul (x, y) piezasRestantes
   |Azul `elem` piezasRestantes = (translated (x+1.5) (y+2.0) (coloured black (text "2")))
   |otherwise = (translated (x+1.5) (y+2.0) (coloured white (text "X"))) & (translated (x+1.5)
(y+2.0) (coloured black (text "2")))
ajustaEnumeracionPiezaColocada Roja (x, y) piezasRestantes
   |Roja `elem` piezasRestantes = (translated (x+1.0) (y+3.5) (coloured black (text "3")))
   |otherwise = (translated (x+1.0) (y+3.5) (coloured white (text "X"))) & (translated (x+1.0)
(y+3.5) (coloured black (text "2")))
ajustaEnumeracionPiezaColocada Naranja (x, y) piezasRestantes
   |Naranja `elem` piezasRestantes = (translated (x+3.0) (y+1.0) (coloured black (text "4")))
   |otherwise = (translated (x+3.0) (y+1.0) (coloured white (text "X"))) & (translated (x+3.0)
(y+1.0) (coloured black (text "4")))
ajustaEnumeracionPiezaColocada Rosa (x, y) piezasRestantes
   |Rosa `elem` piezasRestantes = (translated (x+1.0) (y+3.0) (coloured black (text "5")))
   |otherwise = (translated (x+1.0) (y+3.0) (coloured white (text "X"))) & (translated (x+1.0)
(y+3.0) (coloured black (text "5")))
ajustaEnumeracionPiezaColocada Amarilla (x, y) piezasRestantes
   |Amarilla `elem` piezasRestantes = (translated (x+1.0) (y+0.5) (coloured black (text "6")))
   |otherwise = (translated (x+1.0) (y+0.5) (coloured white (text "X"))) & (translated (x+1.0)
(y+0.5) (coloured black (text "6")))
```

Si la lista de casillas asociada a la pieza está vacía, significa que no está colocada en el tablero y se devuelve blank, es decir no se pinta la pieza porque su lista está vacía y significa que no se ha colocado. En caso contrario, se utiliza una lista por compresión y por cada punto se la lista se pinta un rectángulo con el color asociado a la pieza que se consigue llamando a ***"colorPieza"***

***"pintaPieza"*** se apoya en ***"ajustaEnumeracionPiezaColocada"*** el cual, dependiendo de si la pieza está colocada definitivamente o no, coloca en el centro de la pieza una X blanca. Si la pieza no está dentro de la lista de piezas disponibles, se colocará una X blanca, sino no. Pero justo antes de colocar o no la X blanca, éste método coloca una enumeración (1, 2, 3, 4, 5, 6) dependiendo de la pieza, es decir, un número del uno al seis para cada una de la pieza, para que el jugador sepa en todo momento con cual número se selecciona cada pieza. Para conseguir que la X blanca y la enumeración estén ajustadas en el centro de la pieza, se ha hecho uso del primer

punto de la lista de puntos que conforma la pieza, y se les ha sumado unas cantidades a la coordenadas (x, y) hasta dar con el centro de la pieza.

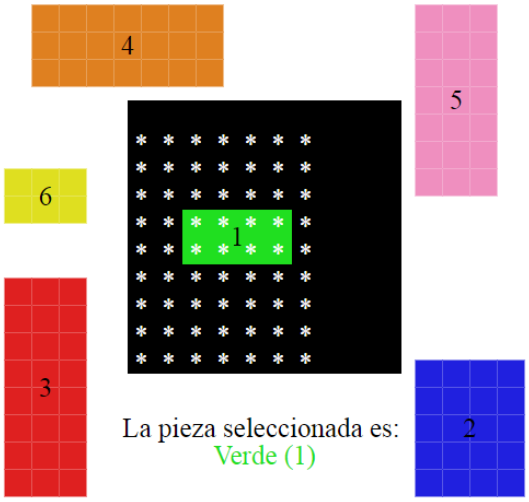



Como podemos observar en la figura de arriba, la enumeración de las piezas se encuentra en el centro de cada pieza, y para el caso de la pieza amarilla, la cual es la única que está colocada definitivamente, la X blanca también se encuentra en todo el centro de la pieza.

```
indicaPiezaSeleccionada :: Pieza -> Picture
indicaPiezaSeleccionada Ninguna =
   scaled 0.85 0.85 (translated (0.0) (-7.0) (coloured black (text "La pieza seleccionada es: "))
& translated (0.0) (-8.0) (coloured black (text "Ninguna")))
indicaPiezaSeleccionada Verde =
   scaled 0.85 0.85 (translated (0.0) (-7.0) (coloured black (text "La pieza seleccionada es: "))
& translated (0.0) (-8.0) (coloured green (text "Verde (1)")))
indicaPiezaSeleccionada Azul =
   scaled 0.85 0.85 (translated (0.0) (-7.0) (coloured black (text "La pieza seleccionada es: "))
& translated (0.0) (-8.0) (coloured blue (text "Azul (2)")))
indicaPiezaSeleccionada Roja =
   scaled 0.85 0.85 (translated (0.0) (-7.0) (coloured black (text "La pieza seleccionada es: "))
& translated (0.0) (-8.0) (coloured red (text "Roja (3)")))
indicaPiezaSeleccionada Naranja =
   scaled 0.85 0.85 (translated (0.0) (-7.0) (coloured black (text "La pieza seleccionada es: "))
& translated (0.0) (-8.0) (coloured orange (text "Naranja (4)")))
indicaPiezaSeleccionada Rosa =
   scaled 0.85 0.85 (translated (0.0) (-7.0) (coloured black (text "La pieza seleccionada es: "))
& translated (0.0) (-8.0) (coloured pink (text "Rosa (5)")))
indicaPiezaSeleccionada Amarilla =
   scaled 0.85 0.85 (translated (0.0) (-7.0) (coloured black (text "La pieza seleccionada es: "))
& translated (0.0) (-8.0) (coloured yellow (text "Amarilla (6)")))
```

Para pintar el texto que señaliza al jugador la pieza que tiene actualmente seleccionada, se hace uso de la siguiente estructura.

El método ***"indicaPiezaSeleccionada"*** recibe una Pieza y, dependiendo de la pieza de la que se trate, devuelve una combinación de 2 pictures, por un lado un text (picture) con la frase en negro: "La pieza seleccionada es: " y, dependiendo del case (Pieza) la continuación de la frase que indica el nombre la pieza en su color.




Como vemos en la imagen, el color de la frase "La pieza seleccionada es: " está en negro, y la continuación depende de la pieza seleccionada, en el caso de la verde pues estará "Verde (1)" en color verde.

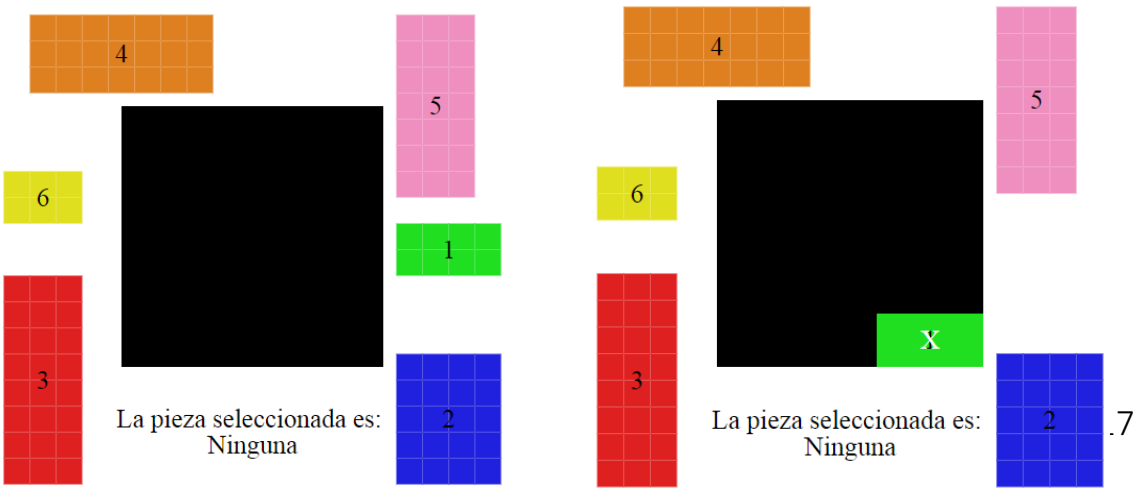
Para dibujar las piezas en los alrededores del tablero, hay que tener en cuenta que el hecho de que se hallen pintadas en los alrededores, significa automáticamente que su lista de puntos asociada en el estado está vacía, es decir, que no está colocada ni de manera temporal ni definitiva en el tablero.

En el momento en que se coloque en el tablero de alguna de esas 2 maneras, la pieza desaparecerá de los alrededores para pasar a colocarse en el tablero.

```
pintaPiezasDisponibles :: [(Pieza, [Casilla])] -> Picture
pintaPiezasDisponibles ls = pictures[if length(coordPieza)==0 then (dibujaPiezaDisponible pieza)
else blank|(pieza, coordPieza)<-ls]

dibujaPiezaDisponible :: Pieza -> Picture
dibujaPiezaDisponible Verde =
   (translated (7.5) (-0.5) (coloured black (text "1"))) &
   pictures[translated ((6.0)+k) ((-1.0)+z) (coloured green (solidRectangle 1 1))|(k, z)<-
(dimensionPieza Verde)]
dibujaPiezaDisponible Azul =
   (translated (7.5) (-7.0) (coloured black (text "2"))) &
   pictures[translated ((6.0)+k) ((-9.0)+z) (coloured blue (solidRectangle 1 1))|(k, z)<-
(dimensionPieza Azul)]
dibujaPiezaDisponible Roja =
   (translated (-8.0) (-5.5) (coloured black (text "3"))) &
   pictures[translated ((-9.0)+k) ((-9.0)+z) (coloured red (solidRectangle 1 1))|(k, z)<-
(dimensionPieza Roja)]
dibujaPiezaDisponible Naranja =
   (translated (-5.0) (7.0) (coloured black (text "4"))) &
   pictures[translated ((-8.0)+k) ((6.0)+z) (coloured orange (solidRectangle 1 1))|(k, z)<-
(dimensionPieza Naranja)]
dibujaPiezaDisponible Rosa =
   (translated (7.0) (5.0) (coloured black (text "5"))) &
   pictures[translated ((6.0)+k) ((2.0)+z) (coloured pink (solidRectangle 1 1))|(k, z)<-
(dimensionPieza Rosa)]
dibujaPiezaDisponible Amarilla =
   (translated (-8.0) (1.5) (coloured black (text "6"))) &
   pictures[translated ((-9.0)+k) ((1.0)+z) (coloured yellow (solidRectangle 1 1))|(k, z)<-
(dimensionPieza Amarilla)]
```

***"pintaPiezaDisponibles"*** recibe una lista de tuplas binarias donde cada tupla contiene la pieza y la lista de puntos donde está construida la pieza en el tablero. Si dicha lista está vacía, significa que no está pintada en el tablero y por tanto si no está pintada en el tablero, tiene que estar pintada en los alrededores para que el jugador pueda visualizar correctamente las piezas de las que dispone. Si sucede lo contrario, es decir que la lista de los puntos asociado a la pieza no está vacía, significa que está colocada en el tablero y por tanto no se ha de pintar en los alrededores.



Como podemos observar en la imagen izquierda, la lista de puntos asociados a la pieza verde del estado, se encuentra vacía y eso significa que no está pintada en el tablero. En la derecha sucede todo lo contrario, la lista de puntos asociada a la pieza verde no está vacía, por tanto ha de estar dibujada en el tablero pero no en los alrededores.

Para pintar los asteriscos, que no son más que la representación de las zonas donde, para una pieza elegida, sí se puede colocar dicha pieza en dichas zonas, se ha utilizado el siguiente método:

```
pintaAsteriscos :: [Casilla] -> Picture
pintaAsteriscos iniciosPieza = pictures[translated x (y-0.15)
(coloured white (text "*"))|(x, y)<-iniciosPieza]
```

***"pintaAsteriscos"*** recibe la lista de puntos donde se puede colocar la pieza, y dichos puntos como esquinas inferiores izquierdas de la pieza, devuelta por el método ***"calculaOpciones"***.

Al método ***"pintaVictoria"*** se le pasa la lista de piezas disponibles, y si ésta está vacía significa que se han podido colocar todas, de manera definitiva, y por tanto se ha rellenado el tablero completo, luego devuelve un text (picture).

```
pintaVictoria :: [Pieza] -> Picture
pintaVictoria [] = scaled 2.0 2.0 (translated 0.0 0.0 (coloured
black (text "Victoria")))
pintaVictoria _ = blank
```


La pieza seleccionada es:
Ninguna

Por último, disponemos del método ***"pintaMundo"*** que representa de manera gráfica es estado en que nos encontramos cada vez que se produce un evento interactivo llevado a cabo por el jugador.

```
pintaMundo :: Mundo -> Picture
pintaMundo (piezasRestantes, piezaElegida, recuadrosLibres, verde, azul,
roja, naranja, rosa, amarilla) =
   (pintaVictoria piezasRestantes) &
   (indicaPiezaSeleccionada piezaElegida) &
   (pintaAsteriscos (calculaOpciones recuadrosLibres piezaElegida)) &
   (pintaPieza Verde verde piezasRestantes) &
   (pintaPieza Azul azul piezasRestantes) &
   (pintaPieza Roja roja piezasRestantes) &
   (pintaPieza Naranja naranja piezasRestantes) &
   (pintaPieza Rosa rosa piezasRestantes) &
   (pintaPieza Amarilla amarilla piezasRestantes) &
   (pintaPiezasDisponibles [(Verde, verde), (Azul, azul), (Roja, roja),
(Naranja, naranja), (Rosa, rosa), (Amarilla, amarilla)]) &
   (pintaTablero (tablero)) -- &
   -- (coordinatePlane)
```

Este método devuelve en un picture que no es más que combinación de un conjunto de picture's devueltos los métodos que hemos ido comentando a lo largo del trabajo hasta llegar aquí. Se pintan en este orden:

Plano 2D -> Tablero negro -> piezas en los alrededores -> piezas colocadas en el tablero -> asteriscos (opciones) -> text "pieza seleccionada: " -> (Si es o no) text "victoria".

## 5: TRATAMIENTO DE LOS EVENTOS INTERACTIVOS

Si la lista de piezas disponibles está vacía, significa que ya hemos llegado a un tablero ganador, y tenemos de devolver el mismo estado con el que entra en ***"resuelveEvento"*** si surge un evento.

```
resuelveEvento :: Event -> Mundo -> Mundo

resuelveEvento _ estado@([], _, _, _, _, _, _, _, _) = estado
```

Si no tenemos ninguna pieza seleccionada, cualquier otra tecla que se pulse que no sea las asociadas a escoger piezas, hará que devuelva el mismo estado que tenemos.

Por otra parte, si la pieza que estamos intentando seleccionar está colocada definitivamente, no se hallará en la lista de piezas disponibles y por tanto devolverá el mismo estado que teníamos.

Si estando la pieza colocada de manera temporal y actualmente no hay ninguna seleccionada, al pulsar otra vez el botón asociado a dicha pieza hará que los puntos de la zona donde estaba colocada, se recuperen sumándose a la lista de vacíos, y como la pieza queda seleccionada, se vuelven a pintar los asteriscos de opciones.

Cualquier otro caso, traduce el evento a la pieza asociada y sería la nueva pieza seleccionada, la cual daría lugar a que se pintasen los asteriscos de opciones para esa nueva pieza

```
resuelveEvento evento estado@(piezasRestantes, Ninguna, recuadrosLibres, verde, azul,
roja, naranja, rosa, amarilla)
   |(not(evento `elem` eventoEligeFicha)) = estado
   |(not((traduceEventoAPieza evento) `elem` piezasRestantes)) = estado
   |(length(verde) /= 0) && ((traduceEvento) == Verde) =
      (piezasRestantes, Verde, recuadrosLibres++verde, verde, azul, roja, naranja,
rosa, amarilla)
   |(length(azul) /= 0) && ((traduceEvento) == Azul) =
      (piezasRestantes, Azul, recuadrosLibres++azul, verde, azul, roja, naranja, rosa,
amarilla)
   |(length(roja) /= 0) && ((traduceEvento) == Roja) =
      (piezasRestantes, Roja, recuadrosLibres++roja, verde, azul, roja, naranja, rosa,
amarilla)
   |(length(naranja) /= 0) && ((traduceEvento) == Naranja) =
      (piezasRestantes, Naranja, recuadrosLibres++naranja, verde, azul, roja, naranja,
rosa, amarilla)
   |(length(rosa) /= 0) && ((traduceEvento) == Rosa) =
      (piezasRestantes, Rosa, recuadrosLibres++rosa, verde, azul, roja, naranja, rosa,
amarilla)
   |(length(amarilla) /= 0) && ((traduceEvento) == Amarilla) =
      (piezasRestantes, Amarilla, recuadrosLibres++amarilla, verde, azul, roja,
naranja, rosa, amarilla)
   |otherwise = (piezasRestantes, (traduceEvento), recuadrosLibres, verde, azul, roja,
naranja, rosa, amarilla)
   where traduceEvento = (traduceEventoAPieza evento)
```

```
traduceEventoAPieza :: Event -> Pieza
traduceEventoAPieza (KeyPress "1") = Verde
traduceEventoAPieza (KeyPress "2") = Azul
traduceEventoAPieza (KeyPress "3") = Roja
traduceEventoAPieza (KeyPress "4") = Naranja
traduceEventoAPieza (KeyPress "5") = Rosa
traduceEventoAPieza (KeyPress "6") = Amarilla

eventoEligeFicha :: [Event]
eventoEligeFicha = [KeyPress "1",
                    KeyPress "2",
```

```
                    KeyPress "3",
                    KeyPress "4",
                    KeyPress "5",
                    KeyPress "6"]
```

Si al hacer **Click Izquierdo**, el cuadro donde lo hacemos no pertenece a la lista de puntos (opciones) que devuelve ***"calculaOpciones"*** para la pieza que tenemos elegida, que son los cuadros a modo de esquina inferior izquierda donde se puede colocar la pieza, devuelve el mismo estado que teníamos.

Si lo anterior si se cumple y pinchamos en cualquiera de las opciones, lo que hace es colocar a modo de ilustración la pieza, guardando en su lista asociada los puntos del plano que forman la pieza, que son construidos truncando la coordenada donde hemos hecho **Click** con el método ***"truncarCoordenadas"***, y utilizándola como esquina inferior izquierda de la pieza que construimos llamando a ***"construirPieza"***.

```
resuelveEvento (MousePress LeftButton (x, y))
estado@(piezasRestantes, piezaElegida, recuadrosLibres, verde, azul,
roja, naranja, rosa, amarilla)
   |not(tC `elem` (calculaOpciones recuadrosLibres piezaElegida)) =
estado
   |piezaElegida == Verde = (piezasRestantes, piezaElegida,
recuadrosLibres, cP, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Azul = (piezasRestantes, piezaElegida,
recuadrosLibres, verde, cP, roja, naranja, rosa, amarilla)
   |piezaElegida == Roja = (piezasRestantes, piezaElegida,
recuadrosLibres, verde, azul, cP, naranja, rosa, amarilla)
   |piezaElegida == Naranja = (piezasRestantes, piezaElegida,
recuadrosLibres, verde, azul, roja, cP, rosa, amarilla)
   |piezaElegida == Rosa = (piezasRestantes, piezaElegida,
recuadrosLibres, verde, azul, roja, naranja, cP, amarilla)
   |piezaElegida == Amarilla = (piezasRestantes, piezaElegida,
recuadrosLibres, verde, azul, roja, naranja, rosa, cP)
   where tC = (truncarCoordenadas (x, y))
         cP = construirPieza tC (dimensionPieza piezaElegida)
```

Si tabulamos con una pieza seleccionada pero sin que esté colocada en ninguna zona habiendo hecho previamente **Click Izquierdo**, es decir teniendo la lista de puntos de la pieza asociada a vacío, devuelve el estado anterior. Si no tuviésemos esta estructura, lo que provocaría es que al darle a **Ctrl** se deseleccionase la pieza por culpa de la estructura que explicaremos en el apartado E, y no queremos que esto pase porque para deseleccionar está pensado la tecla escape que explicaremos en el apartado H.

```
resuelveEvento (KeyPress "Ctrl") estado@(piezasRestantes,
piezaElegida, recuadrosLibres, verde, azul, roja, naranja, rosa,
amarilla)
    |piezaElegida == Verde && length(verde) == 0 = estado
    |piezaElegida == Azul && length(azul) == 0 = estado
    |piezaElegida == Roja && length(roja) == 0 = estado
    |piezaElegida == Naranja && length(naranja) == 0 = estado
    |piezaElegida == Rosa && length(rosa) == 0 = estado
    |piezaElegida == Amarilla && length(amarilla) == 0 = estado
```

Si pulsamos **Ctrl** para una pieza elegida y cuya lista de puntos asociado no está vacía, significa que está ilustrada en el tablero, es decir, que está construido porque anteriormente hicimos **Click Izquierdo** en uno de los asteriscos y se ha generó la pieza, luego al pulsar el **Ctrl** lo que hacemos es colocarla de manera temporal. La pieza elegida pasa a valer “Ninguna”, y el valor de la lista de puntos asociado no cambia, es la misma que teníamos antes que contenía los puntos en el tablero para poder pintarlo.

```
resuelveEvento (KeyPress "Ctrl") estado@(piezasRestantes, piezaElegida,
recuadrosLibres, verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Verde = (piezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres verde), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Azul = (piezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres azul), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Roja = (piezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres roja), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Naranja = (piezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres naranja), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Rosa = (piezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres rosa), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Amarilla = (piezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres amarilla), verde, azul, roja, naranja, rosa, amarilla)
```

Si pulsamos **Enter** teniendo una pieza elegida, pero sin estar ilustrada en ninguna zona del tablero, es decir, la lista de puntos asociada a la pieza está vacía, devuelve el estado anterior. Esto se hace para evitar que se trague la pieza por culpa de la estructura del apartado G

```
resuelveEvento (KeyPress "Enter") estado@(piezasRestantes,
piezaElegida, recuadrosLibres, verde, azul, roja, naranja, rosa,
amarilla)
    |piezaElegida == Verde && length(verde) == 0 = estado
    |piezaElegida == Azul && length(azul) == 0 = estado
    |piezaElegida == Roja && length(roja) == 0 = estado
    |piezaElegida == Naranja && length(naranja) == 0 = estado
    |piezaElegida == Rosa && length(rosa) == 0 = estado
    |piezaElegida == Amarilla && length(amarilla) == 0 = estado
```

Si pulsamos **enter** teniendo una pieza seleccionada y cuya lista de puntos asociado no está vacía, es decir, la pieza está ilustrada, lo que ocurre es que hemos colocado de manera definitiva una pieza, luego se elimina de la lista de piezas disponibles mediante el método ***"quitaPieza"***, y los puntos de la zona ocupada hay que quitárselos a la lista de vacíos llamando al método ***"quitaRecuadrosLibres"***. Para obtener la lista de puntos La pieza seleccionada pasa a ser "Ninguna".

```
resuelveEvento (KeyPress "Enter") estado@(piezasRestantes, piezaElegida,
recuadrosLibres, verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Verde = (nuevasPiezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres verde), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Azul = (nuevasPiezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres azul), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Roja = (nuevasPiezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres roja), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Naranja = (nuevasPiezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres naranja), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Rosa = (nuevasPiezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres rosa), verde, azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Amarilla = (nuevasPiezasRestantes, Ninguna, (quitaRecuadrosLibres
recuadrosLibres amarilla), verde, azul, roja, naranja, rosa, amarilla)
   where nuevasPiezasRestantes = (quitaPieza piezasRestantes piezaElegida)
```

Pulsando escape, si había una pieza seleccionada y ya estuviese reflejada en el tablero como si no lo estaba, devuelve un nuevo estado con la lista asociada a dicha pieza vacía y la pieza elegida a "Ninguno", ergo quita la ilustración de la pieza, si estaba, quita los asteriscos y devuelve la pieza a los alrededores del tablero.

```
resuelveEvento (KeyPress "Esc") estado@(piezasRestantes,
piezaElegida, recuadrosLibres, verde, azul, roja, naranja, rosa,
amarilla)
   |piezaElegida == Verde = (piezasRestantes, Ninguna,
recuadrosLibres, [], azul, roja, naranja, rosa, amarilla)
   |piezaElegida == Azul = (piezasRestantes, Ninguna,
recuadrosLibres, verde, [], roja, naranja, rosa, amarilla)
   |piezaElegida == Roja = (piezasRestantes, Ninguna,
recuadrosLibres, verde, azul, [], naranja, rosa, amarilla)
   |piezaElegida == Naranja = (piezasRestantes, Ninguna,
recuadrosLibres, verde, azul, roja, [], rosa, amarilla)
   |piezaElegida == Rosa = (piezasRestantes, Ninguna,
recuadrosLibres, verde, azul, roja, naranja, [], amarilla)
   |piezaElegida == Amarilla = (piezasRestantes, Ninguna,
recuadrosLibres, verde, azul, roja, naranja, rosa, [])
```

## 6: SOLUCIONES

```
data Pieza = Ninguna | Verde | Azul | Roja | Naranja | Rosa | Amarilla
deriving (Show, Eq)
data Casilla = Vacia | C (Double, Double) deriving (Show, Eq)

type Movimiento = (Pieza,Casilla)
type Estado = ([Pieza], Pieza, [Casilla], [Movimiento])

piezas :: [Pieza]
piezas = [Verde,Azul,Roja,Naranja,Rosa,Amarilla]

dimensionPieza :: Pieza -> [Casilla]
dimensionPieza Verde = [C (x, y)|x<-[0.0..3.0], y<-[0.0..1.0]]
dimensionPieza Azul = [C (x, y)|x<-[0.0..3.0], y<-[0.0..4.0]]
dimensionPieza Roja = [C (x, y)|x<-[0.0..2.0], y<-[0.0..7.0]]
dimensionPieza Naranja = [C (x, y)|x<-[0.0..6.0], y<-[0.0..2.0]]
dimensionPieza Rosa = [C (x, y)|x<-[0.0..2.0], y<-[0.0..6.0]]
dimensionPieza Amarilla = [C (x, y)|x<-[0.0..2.0], y<-[0.0..1.0]]

tablero :: [Casilla]
tablero = [C (((-5.0)+1/2)+x,((-5.0)+1/2)+y)|x<-[0.0..9.0], y<-[0.0..9.0]]

calculaOpciones :: [Casilla] -> Pieza -> [Movimiento]
calculaOpciones _ Ninguna = []
calculaOpciones recuadrosLibres piezaElegida =
   [(piezaElegida, puntoPlano) | puntoPlano <- recuadrosLibres,
(calculaOpcionesAux puntoPlano (dimensionPieza piezaElegida)
recuadrosLibres)]

calculaOpcionesAux :: Casilla -> [Casilla] -> [Casilla] -> Bool
calculaOpcionesAux (C (x, y)) dimension recuadrosLibres = and[(C (x+k, y+z))
`elem` recuadrosLibres|(C (k, z)) <- dimension]

construirPieza :: Casilla -> [Casilla] -> [Casilla]
construirPieza (C (x, y)) dimension = [(C (x+k, y+z))|(C (k, z))<-dimension]

quitaPieza :: [Pieza] -> Pieza -> [Pieza]
quitaPieza [x] pieza = []
quitaPieza (x:xs) pieza
   |pieza == x = xs
   |otherwise = x:(quitaPieza xs pieza)

quitaRecuadrosLibres :: [Casilla] -> [Casilla] -> [Casilla]
quitaRecuadrosLibres recuadrosLibres [] = recuadrosLibres
quitaRecuadrosLibres recuadrosLibres (x:xs) = (quitaRecuadrosLibres
(quitaRecuadrosLibresAux recuadrosLibres x) xs)

quitaRecuadrosLibresAux :: [Casilla] -> Casilla -> [Casilla]
quitaRecuadrosLibresAux [x] recuadroLibre = []
quitaRecuadrosLibresAux (x:xs) recuadroLibre
   |x == recuadroLibre = xs
   |otherwise = x:(quitaRecuadrosLibresAux xs recuadroLibre)
```

Se ha definido movimiento como una tupla de (Pieza, Casilla), donde el primer término es la pieza que se coloca y el segundo término será la esquina inferior izquierda de la pieza que colocamos

Casilla se ha cambiado de Type a Data, y ahora puede tomar el valor de Vacía, y el tipo estado contendrá las piezas restantes, la pieza elegida, casillas libres y los movimientos que se van haciendo.

Ese 4º valor se inicializará a (Ninguna, Vacía).

## 7: BÚSQUEDA DE TODOS LOS TABLEROS SATISFACTORIOS

```
calculaSiguientes :: Estado -> [[Movimiento]]
calculaSiguientes estado@([], _, [], secuencia) = [secuencia]
calculaSiguientes estado@(_, piezaElegida, recuadrosLibres, _) =
   concat[(recursivo opcion estado)|opcion<-(calculaOpciones recuadrosLibres piezaElegida)]

recursivo :: Movimiento -> Estado -> [[Movimiento]]
recursivo (pieza, inicioPieza) estado@(piezasRestantes, _, recuadrosLibres, secuencia)
   |length(nuevasPiezasRestantes)==0 = calculaSiguientes (nuevasPiezasRestantes, Ninguna,
nuevosRecuadrosLibres, secuencia++[(pieza, inicioPieza)])
   |otherwise = calculaSiguientes (nuevasPiezasRestantes, nuevasPiezasRestantes!!0,
nuevosRecuadrosLibres, secuencia++[(pieza, inicioPieza)])
   where nuevasPiezasRestantes = (quitaPieza piezasRestantes pieza)
         nuevosRecuadrosLibres = quitaRecuadrosLibres recuadrosLibres (construirPieza inicioPieza
(dimensionPieza pieza))
```

Genera todos los tableros posibles, tanto los que se quedan corto de opciones y la lista por compresión devuelve lista vacía, como los que llegan hasta el final rellenando todo el tablero, y devuelve las 36 soluciones del tablero que existen.

# 8: BÚSQUEDA DE ESPACIOS DE ESTADO POR ANCHURA

```
esEstadoFinal :: Estado -> Bool
esEstadoFinal estado@([], _, [], _) = True
esEstadoFinal _ = False

aplicables :: Estado -> [Movimiento]
aplicables (_, pieza, recuadrosLibres, _) = (calculaOpciones recuadrosLibres pieza)

aplica :: Movimiento -> Estado -> Estado
aplica (pieza, inicioPieza) (piezasRestantes, _, recuadrosLibres, secuencia)
   |length(nuevasPiezasRestantes) == 0 = (nuevasPiezasRestantes, Ninguna,
nuevosRecuadrosLibres, secuencia++[(pieza, inicioPieza)])
   |otherwise = (nuevasPiezasRestantes, nuevasPiezasRestantes!!0,
nuevosRecuadrosLibres, secuencia++[(pieza, inicioPieza)])
   where nuevasPiezasRestantes = (quitaPieza piezasRestantes pieza)
         nuevosRecuadrosLibres = quitaRecuadrosLibres recuadrosLibres (construirPieza
inicioPieza (dimensionPieza pieza))

busquedaAnchura :: Estado -> Maybe [Movimiento]
busquedaAnchura estado = metodo1 [] [estado]

metodo1 :: [Estado] -> [Estado] -> Maybe [Movimiento]
metodo1 explorados frontera
  |length(frontera) == 0 = Nothing
  |otherwise = metodo2 (frontera!!0) (explorados) (tail frontera)

metodo2 :: Estado -> [Estado] -> [Estado] -> Maybe [Movimiento]
metodo2 estado@(_, _, _, sucesion) explorados frontera
   |esEstadoFinal estado == True = Just sucesion
   |otherwise = metodo1 (explorados++[estado]) (anyadirSucesoresFrontera estado
explorados frontera)

anyadirSucesoresFrontera :: Estado -> [Estado] -> [Estado] -> [Estado]
anyadirSucesoresFrontera estado explorados frontera =
   anyadirFrontera [aplica opcion estado|opcion<-(aplicables estado)] explorados
frontera

anyadirFrontera :: [Estado] -> [Estado] -> [Estado] -> [Estado]
anyadirFrontera [] explorados frontera = frontera
anyadirFrontera (x:xs) explorados frontera
   |(not(x `elem` explorados)) && (not(x `elem` frontera)) == True = (anyadirFrontera
xs explorados (frontera++[x]))
   |otherwise = frontera
```

Llamamos a metodo1 con ***"explorados"*** inicializado a vacío y ***"frontera"*** inicializada con un estado, el primero con el que se instancia a ***"busquedaAnchura"***. Si ***"frontera"*** se vacía y no hemos encontrado el estado final, devuelve ***"Nothing"***, caso contrario llamo a metodo2 sacando de ***"frontera"*** el primer elemento por la izquierda (FIF0). Si dicho estado es final, he terminado y en caso contrario saco sus sucesores que no estén

ni en ***"frontera"*** ni en ***"explorados"***, los meto por la derecha de la ***"frontera"*** (por la derecha) y el estado del que he sacado los sucesores lo meto en ***"explorados"***, y vuelvo a sacar uno por la izquierda de la ***"frontera"***… así sucesivamente hasta que encuentre un estado que al sacarlo de la "frontera" sea final.

```
{--
calculaSiguientes estadoInicial1

[[(Ninguna,Vacia),(Verde,C (-4.5,-4.5)),(Azul,C (-4.5,-2.5)),(Roja,C
(2.5,-4.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-0.5,-4.5)),(Amarilla,C
(2.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-4.5,-4.5)),(Azul,C (-4.5,-2.5)),(Roja,C
(2.5,-2.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-0.5,-4.5)),(Amarilla,C
(2.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-4.5,-4.5)),(Azul,C (-1.5,-2.5)),(Roja,C
(-4.5,-2.5)),(Naranja,C (-1.5,2.5)),(Rosa,C (2.5,-4.5)),(Amarilla,C
(-0.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-4.5,-1.5)),(Azul,C (-4.5,0.5)),(Roja,C
(2.5,-4.5)),(Naranja,C (-4.5,-4.5)),(Rosa,C (-0.5,-1.5)),(Amarilla,C
(2.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-4.5,-1.5)),(Azul,C (-4.5,0.5)),(Roja,C
(2.5,-2.5)),(Naranja,C (-4.5,-4.5)),(Rosa,C (-0.5,-1.5)),(Amarilla,C
(2.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-4.5,0.5)),(Azul,C (-4.5,-4.5)),(Roja,C
(2.5,-4.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-0.5,-4.5)),(Amarilla,C
(2.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-4.5,0.5)),(Azul,C (-4.5,-4.5)),(Roja,C
(2.5,-2.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-0.5,-4.5)),(Amarilla,C
(2.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-4.5,3.5)),(Azul,C (-4.5,-1.5)),(Roja,C
(2.5,-4.5)),(Naranja,C (-4.5,-4.5)),(Rosa,C (-0.5,-1.5)),(Amarilla,C
(2.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-4.5,3.5)),(Azul,C (-4.5,-1.5)),(Roja,C
(2.5,-2.5)),(Naranja,C (-4.5,-4.5)),(Rosa,C (-0.5,-1.5)),(Amarilla,C
(2.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-4.5,3.5)),(Azul,C (-1.5,-1.5)),(Roja,C
(-4.5,-4.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C (2.5,-1.5)),(Amarilla,C
(-0.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,-4.5)),(Azul,C (-1.5,-2.5)),(Roja,C
(-4.5,-4.5)),(Naranja,C (-1.5,2.5)),(Rosa,C (2.5,-4.5)),(Amarilla,C
(-4.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,-4.5)),(Azul,C (-1.5,-2.5)),(Roja,C
(-4.5,-2.5)),(Naranja,C (-1.5,2.5)),(Rosa,C (2.5,-4.5)),(Amarilla,C
(-4.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,-4.5)),(Azul,C (-1.5,-2.5)),(Roja,C
(2.5,-4.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-4.5,-4.5)),(Amarilla,C
(2.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,-4.5)),(Azul,C (-1.5,-2.5)),(Roja,C
(2.5,-2.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-4.5,-4.5)),(Amarilla,C
(2.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,-1.5)),(Azul,C (-1.5,0.5)),(Roja,C
(-4.5,-4.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C (2.5,-1.5)),(Amarilla,C
(-4.5,3.5))],
```

```
[(Ninguna,Vacia),(Verde,C (-1.5,-1.5)),(Azul,C (-1.5,0.5)),(Roja,C
(-4.5,-2.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C (2.5,-1.5)),(Amarilla,C
(-4.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,-1.5)),(Azul,C (-1.5,0.5)),(Roja,C
(2.5,-4.5)),(Naranja,C (-4.5,-4.5)),(Rosa,C (-4.5,-1.5)),(Amarilla,C
(2.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,-1.5)),(Azul,C (-1.5,0.5)),(Roja,C
(2.5,-2.5)),(Naranja,C (-4.5,-4.5)),(Rosa,C (-4.5,-1.5)),(Amarilla,C
(2.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,0.5)),(Azul,C (-1.5,-4.5)),(Roja,C
(-4.5,-4.5)),(Naranja,C (-1.5,2.5)),(Rosa,C (2.5,-4.5)),(Amarilla,C
(-4.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,0.5)),(Azul,C (-1.5,-4.5)),(Roja,C
(-4.5,-2.5)),(Naranja,C (-1.5,2.5)),(Rosa,C (2.5,-4.5)),(Amarilla,C
(-4.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,0.5)),(Azul,C (-1.5,-4.5)),(Roja,C
(2.5,-4.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-4.5,-4.5)),(Amarilla,C
(2.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,0.5)),(Azul,C (-1.5,-4.5)),(Roja,C
(2.5,-2.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-4.5,-4.5)),(Amarilla,C
(2.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,3.5)),(Azul,C (-1.5,-1.5)),(Roja,C
(-4.5,-4.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C (2.5,-1.5)),(Amarilla,C
(-4.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,3.5)),(Azul,C (-1.5,-1.5)),(Roja,C
(-4.5,-2.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C (2.5,-1.5)),(Amarilla,C
(-4.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,3.5)),(Azul,C (-1.5,-1.5)),(Roja,C
(2.5,-4.5)),(Naranja,C (-4.5,-4.5)),(Rosa,C (-4.5,-1.5)),(Amarilla,C
(2.5,3.5))],
[(Ninguna,Vacia),(Verde,C (-1.5,3.5)),(Azul,C (-1.5,-1.5)),(Roja,C
(2.5,-2.5)),(Naranja,C (-4.5,-4.5)),(Rosa,C (-4.5,-1.5)),(Amarilla,C
(2.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (1.5,-4.5)),(Azul,C (-1.5,-2.5)),(Roja,C
(2.5,-2.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-4.5,-4.5)),(Amarilla,C
(-1.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (1.5,-4.5)),(Azul,C (1.5,-2.5)),(Roja,C (-
4.5,-4.5)),(Naranja,C (-1.5,2.5)),(Rosa,C (-1.5,-4.5)),(Amarilla,C
(-4.5,3.5))],
[(Ninguna,Vacia),(Verde,C (1.5,-4.5)),(Azul,C (1.5,-2.5)),(Roja,C (-
4.5,-2.5)),(Naranja,C (-1.5,2.5)),(Rosa,C (-1.5,-4.5)),(Amarilla,C
(-4.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (1.5,-1.5)),(Azul,C (1.5,0.5)),(Roja,C (-
4.5,-4.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C (-1.5,-1.5)),(Amarilla,C
(-4.5,3.5))],
[(Ninguna,Vacia),(Verde,C (1.5,-1.5)),(Azul,C (1.5,0.5)),(Roja,C (-
4.5,-2.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C (-1.5,-1.5)),(Amarilla,C
(-4.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (1.5,0.5)),(Azul,C (1.5,-4.5)),(Roja,C (-
4.5,-4.5)),(Naranja,C (-1.5,2.5)),(Rosa,C (-1.5,-4.5)),(Amarilla,C
(-4.5,3.5))],
[(Ninguna,Vacia),(Verde,C (1.5,0.5)),(Azul,C (1.5,-4.5)),(Roja,C (-
4.5,-2.5)),(Naranja,C (-1.5,2.5)),(Rosa,C (-1.5,-4.5)),(Amarilla,C
(-4.5,-4.5))],
[(Ninguna,Vacia),(Verde,C (1.5,3.5)),(Azul,C (-1.5,-1.5)),(Roja,C
(2.5,-4.5)),(Naranja,C (-4.5,-4.5)),(Rosa,C (-4.5,-1.5)),(Amarilla,C
(-1.5,3.5))],
```

```
[(Ninguna,Vacia),(Verde,C (1.5,3.5)),(Azul,C (1.5,-1.5)),(Roja,C (-
4.5,-4.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C (-1.5,-1.5)),(Amarilla,C
(-4.5,3.5))],
[(Ninguna,Vacia),(Verde,C (1.5,3.5)),(Azul,C (1.5,-1.5)),(Roja,C (-
4.5,-2.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C (-1.5,-1.5)),(Amarilla,C
(-4.5,-4.5))]]

length(calculaSiguientes estadoInicial1)

36
--}

{--
*Main> busquedaAnchura estadoInicial1
Just [(Ninguna,Vacia),(Verde,C (-4.5,-4.5)),(Azul,C (-4.5,-
2.5)),(Roja,C (2.5,-4.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-0.5,-
4.5)),(Amarilla,C (2.5,3.5))]

*Main> busquedaAnchura estadoInicial2
Just [(Ninguna,Vacia),(Azul,C (-4.5,-4.5)),(Verde,C (-
4.5,0.5)),(Roja,C (2.5,-4.5)),(Naranja,C (-4.5,2.5)),(Rosa,C (-0.5,-
4.5)),(Amarilla,C (2.5,3.5))]

*Main> busquedaAnchura estadoInicial3
Just [(Ninguna,Vacia),(Roja,C (-4.5,-4.5)),(Verde,C (-
4.5,3.5)),(Azul,C (-1.5,-1.5)),(Naranja,C (-1.5,-4.5)),(Rosa,C
(2.5,-1.5)),(Amarilla,C (-0.5,3.5))]

*Main> busquedaAnchura estadoInicial4
Just [(Ninguna,Vacia),(Naranja,C (-4.5,-4.5)),(Verde,C (-4.5,-
1.5)),(Azul,C (-4.5,0.5)),(Roja,C (2.5,-4.5)),(Rosa,C (-0.5,-
1.5)),(Amarilla,C (2.5,3.5))]

*Main> busquedaAnchura estadoInicial5
Just [(Ninguna,Vacia),(Rosa,C (-4.5,-4.5)),(Verde,C (-1.5,-
4.5)),(Azul,C (-1.5,-2.5)),(Roja,C (2.5,-4.5)),(Naranja,C (-
4.5,2.5)),(Amarilla,C (2.5,3.5))]

*Main> busquedaAnchura estadoInicial6
Just [(Ninguna,Vacia),(Amarilla,C (-4.5,-4.5)),(Verde,C (-1.5,-
4.5)),(Azul,C (-1.5,-2.5)),(Roja,C (-4.5,-2.5)),(Naranja,C (-
1.5,2.5)),(Rosa,C (2.5,-4.5))]

--}
```