# Bridges

# Roberto Hueso Gómez

# 1 Descripción

El puzle consiste en varias islas que se pueden conectar entre ellas por puentes. Dadas dos islas, se pueden conectar entre ellas por 0, 1 o 2 puentes. Cada isla tiene asociado un número  $x \in \mathbb{N}$  y cada vez que conectamos un puente a esa isla  $x_n = x_{n-1} - 1$  el objetivo es hacer que  $x_n = 0$  en todas las islas. Cuando cumplimos el objetivo, hemos ganado la partida.

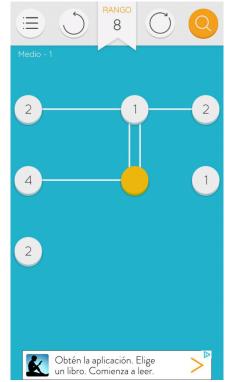


Figura 1: Captura de pantalla durante el juego.

# 2 Representación

Todo el problema se ha representado como un grafo dirigido en el que los vértices son las islas y las aristas los puentes.

Obviaré los detalles del código puesto que para algo detallado es mas simple leer el código.

#### 2.1 Grafo

El grafo se define como

```
data Graph = Empty | G vertice [vertices_conectados] (Graph)
```

Dónde vertice se define como

```
data Vertex = V etiqueta (x, y)
```

Donde la *etiqueta* es el número asociado a la isla y (x,y) son las coordenadas x,y en el plano euclideo de la isla. El campo *vertices\_conectados* indican los vertices a los que se conecta *vertice* de manera que queda representado la arista dirigida *vertice—vertice\_conectado\_1* 

El tipo arista es puramente auxiliar y se define como

```
type Edge = (vertice_origen, vertice_destino)
```

#### 2.2 Juego

He decidido que la *etiqueta* de los nodos sea un entero (los puentes restantes) por simplicidad. Los nodos estarán en puntos discretos del plano, por lo que he decidido que las *coordenadas* se expresen como duplas de enteros.

```
type Coordinates = (Int, Int)
```

El tipo *Mundo* también será una dupla pero en este caso será entre un grafo y unas coordenadas (como método auxiliar) que representan las coordenadas de origen de la arista. Las coordenadas de destino las tomaremos en el manejador de acciones.

```
type Mundo = (Graph, Coordinates)
```

Para simplificar el código al crear los grafos haré las coordenadas (x,y) de los nodos sean múltiplos de 3 partiendo del origen (0,0). También haré que los puentes no se puedan cruzar para evitar tener que comprobar la planaridad del grafo.

#### 2.3 Búsqueda de solución en anchura

El tipo Estado quedará definido como

```
type Estado = Graph
```

Dónde *Graph* representa el estado actual del mapa de islas.

Los movimientos que podremos aplicar a cada situación del mapa vendrán determinados por *Movimiento* que se define como

dónde cada dirección representa desplazarse en esa dirección 3 unidades del plano euclideo desde el *vertice*.

# 3 Código

### 3.1 Juego Interactivo

```
pintaMundo :: Mundo -> Picture
```

Se usa para pintar el mundo trás cada interacción del usuario, solo es necesario usarlo en la definición de la interacción.

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

Se usa para manejar cada interacción del usuario, ya sea con el ratón o con el teclado solo es necesario usarlo en la definición de la interacción. Contempla las interacciones

- Click izquierdo para definir el origen de una arista.
- Click central para definir el destino de una arista.
- Tecla *Escape* para reiniciar el juego a su estado inicial.

#### 3.2 Búsqueda de solución en anchura

```
esEstadoFinal :: Estado -> Bool
```

Se usa en cada estado que se alcanza en el arbol de búsqueda para determinar si se ha llegado a una solución.

```
aplicables :: Estado -> [Movimiento]
```

Se usa para generar las acciones aplicables a cada estado dentro de la búsqueda BFS.

```
aplica :: Movimiento -> Estado -> Estado
```

Para cada acción aplicable (Cuando generamos el arbol de búsqueda) esta función se encarga de generar el nuevo estado al que se llega tras haberla aplicado.

#### 3.3 Ejemplos

Ejecutar resolución en CodeWorld

```
:1 Bridges.hs
main
-- Para probar otros niveles cambiar la constante "main_game"

Ejecutar y visualizar BFS
:1 Bridges_Solver.hs
```

## **4 Referencias**

• Programación declarativa - US<sup>1</sup>

dibuja\_resultado \$ busqueda estado1

- Wikipedia BFS<sup>2</sup>
- Aprende Haskell por el bien de todos³
- Haskell reference<sup>4</sup>

<sup>1</sup>https://www.cs.us.es/cursos/pd/

<sup>&</sup>lt;sup>2</sup>https://en.wikipedia.org/wiki/Breadth-first\_search

<sup>3</sup>http://aprendehaskell.es/main.html

<sup>4</sup>http://zvon.org/other/haskell/Outputglobal/index.html

- CodeWorld reference<sup>5</sup>
- Haskell Text Package<sup>6</sup>
- Haskell List Package<sup>7</sup>

<sup>&</sup>lt;sup>5</sup>https://code.world/doc-haskell/CodeWorld.html

<sup>6</sup>https://hackage.haskell.org/package/text-1.2.3.0/docs/Data-Text.html

<sup>7</sup>https://hackage.haskell.org/package/base-4.10.1.0/docs/Data-List.html