2 | Estados y espacio de búsqueda

Rodrigo Eduardo Colín Rivera
Verónica Esther Arriola Ríos

Objetivo

Que el alumno comprenda la noción de *Estado* y *Espacio de Búsqueda*, y que pueda representar en una estructura de datos todos los posibles estados de un mundo dado.

Introducción

Dado un problema o modelo es posible determinar las características del mundo descrito en él. Uno de los puntos a tratar es la representación de dicho mundo y el concepto de estado. Un estado es una configuración posible del mundo con el que se trabaja. Por ejemplo, un juego de ajedrez se compone de un tablero de 64 cuadros donde inicialmente están 32 piezas (16 para cada jugador). El estado inicial del ajedrez se muestra en la Figura 2.1.

Cuando una pieza se mueve, el estado del ajedrez cambia; pues la posición y cantidad de las piezas en juego determinan el estado del ajedrez. De esta manera el *espacio de estados* se representa como una gráfica donde cada nodo representa un estado del problema y las aristas que los unen son la aplicación de un operador o función.

Es decir, el espacio de estados son todas las posibles configuraciones del problema. En el caso del ajedrez, las operaciones que unen cada estado en la gráfica representan el movimiento de una pieza por parte del jugador y son el otro elemento requerido para el definir el *sistema de transiciones de estados*.

[1]Milton A. Ramírez Klapp, Notas de Inteligencia Artificial, Universidad San Sebastián, Facultad de Ingeniería y Tecnología.

Figura 2.1 Estado inicial del juego de ajedrez.

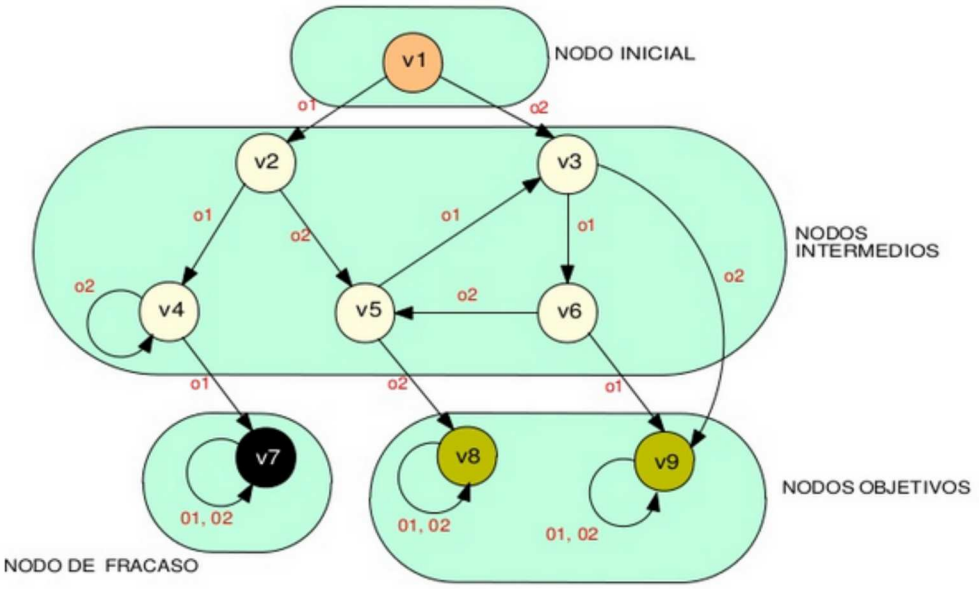

Figura 2.2 Ejemplo de un espacio de estados con las transiciones entre ellos. [1]

Figura 2.3 Ejemplo de un juego de Gato donde gana el jugador «X». [2]

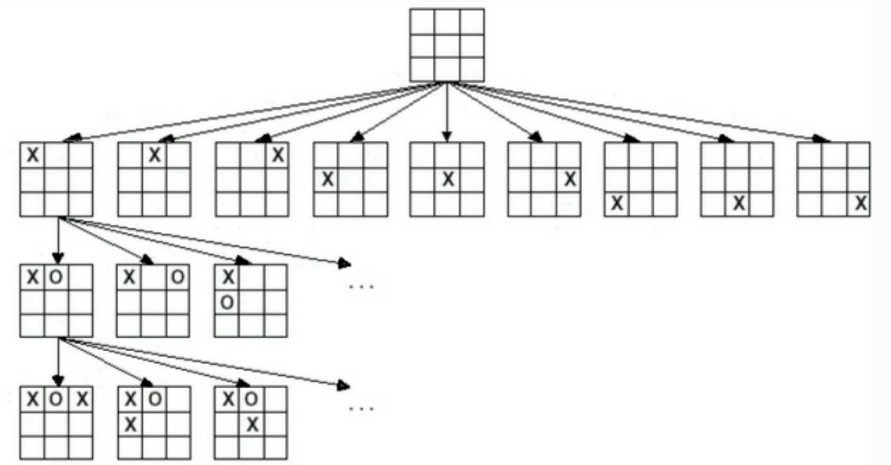

Figura 2.4 Espacio de búsqueda en el juego del Gato representado como un árbol.

Desarrollo e implementación

La práctica consiste de generar los sucesores de los estados válidos del juego del Gato (también llamado Tres en línea).

Recordemos que el juego del Gato es entre dos jugadores, representados por los símbolos «X» y «O» que toman turnos para marcar los espacios de un tablero de 3x3. Un jugador gana cuando logra tener una línea, ya sea horizontal, vertical o diagonal, de 3 símbolos correspondientes.

Afortunadamente el juego del Gato es lo bastante simple como para evitar ciclarse al generar el espacio de estados y eso es una característica importante que ayuda a la construcción del espacio de estados, ya que podemos hacer uso de un árbol (que es un tipo especial de gráfica) para almacenar los estados.

Por tanto, se necesitan los siguientes elementos para representar el espacio de estados del Gato:

1. **Representación del estado**

   Una manera de representar el estado del Gato, se usará la siguiente idea:

[2] Autor: Gdr (http://en.wikipedia.org/wiki/User:Gdr)

```
int[][] tablero = [[0,0,1], [0,0,0], [4,4,0]]
```

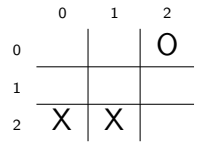

Figura 2.5 Representación gráfica del estado del Gato

Con 1 representando al primer jugador y 4 al segundo (fig 2.5).

2. **Función generadora**
   Se necesitará implementar una función que genere los sucesores de un estado del juego del Gato, considerando sólo las jugadas válidas y descartando simetrías.

3. **Función de comprobación**
   Se necesitará implementar una función que verifique si hubo ganador en un estado dado, utilizando una bandera para indicar que no se debe generar sucesores de ese estado, para evitar generar estados inalcanzables.

Implementación

Se volverá a trabajar con `Processing`.

Se debe programar lo referente a generación de sucesores de un estado, verificación de simetrías y agregar variables que lleven el conteo de empates y juegos ganados por cada jugador, esto se imprimirá junto con la información de nodos del nivel generado.

En pocas palabras es necesario implementar los siguientes métodos:

- `LinkedList<Gato> generaSucesores()`
- `boolean esSimetricoDiagonalInvertida(Gato otro)`
- `boolean esSimetricoDiagonal(Gato otro)`
- `boolean esSimetricoVerticalmente(Gato otro)`
- `boolean esSimetricoHorizontalmente(Gato otro)`
- `boolean esSimetrico90(Gato otro)`
- `boolean esSimetrico180(Gato otro)`
- `boolean esSimetrico270(Gato otro)`

Cada método se encuentra especificado dentro del archivo `Gatos.java`.

Punto extra

Si lo desean pueden extenderse e implementar una función de dispersión (*hash*) para cada estado generado, e implementar una lista cerrada. De esta manera se evita expandir rutas que ya se habían expandido anteriormente. Podrán obtener hasta **2** puntos extras pero cuidado: ¡no es nada trivial completar este ejercicio! pues gatos que son equivalentes bajo simetrías deben ser mapeados al mismo código de dispersión. No se sorprendan si no pueden resolver el ejercicio a la perfección.

Requisitos y resultados

Para evaluar y calificar la práctica es necesario que se implementen todos los métodos mencionados e indicados en el código, respetando implementar sólo lo que se pide (para evitar comportamientos extraños de la simulación). Es completamente válido utilizar bibliotecas adicionales si lo consideran necesario, así como la creación y uso de sus propios métodos auxiliares si lo desean.

Deben correr su simulación de los espacios de estados del Gato sin simetrías, y después con ellas. Agreguen en su archivo `readme` un pequeño párrafo detallando sus observaciones con respecto a lo anterior.

Si crean métodos auxiliares, no olviden documentar cuál es su función.