# Librería del libro de Russell & Norvig : AIMA

- Qué contiene la librería AIMA?
  - Un Framework (clases genéricas Java) que separa:
    - La representación del problema de…
    - Los algoritmos de búsqueda (carpeta “Search”)
  - Demos de varios problemas clásicos: Reinas, 8Puzzle,…
- Qué búsquedas implementa?
  - Búsqueda ciega (uninformed): Profundidad limitada, Profundidad iterativa y otras
  - Búsqueda heurística (informed): A*, Hill Climbing, Simulated Annealing



# Librería del libro de Russell & Norvig : AIMA

- Cómo se consigue?
  - Usamos la 2ªedición del libro: http://aima.cs.berkeley.edu/2nd-ed/)
    - aima-java0.95.3rdOct2009.zip Fuentes de la Librería con las clases AIMA
- Qué hay en el material en el Campus Virtual:
  - Ejemplo 8-puzzle: ejemplo.rar
    - \ej8p  Carpeta con lo necesario para ejecutar y crear ejemplos con AIMA
    - Cómo se instala? => proyecto eclipse
  - aima-2.jar  Librería con las clases AIMA
  - aima-java0.95.3rdOct2009.zip

## Convenio de representación PROLOG: Elementos y Pasos

1. Definir componentes de un estado: `estado(...)`
2. Estado inicial `inicial(Estado)`

   Estados objetivos `objetivo(Estado) :- condicionesObjetivo.`

   Estados de peligro `peligro(Estado) :- CondiciónPeligro.`
3. Operadores (o función sucesor o movimiento):
   - Acciones disponibles para pasar de un estado al siguiente:

     `movimiento(Estado, EstadoSiguiente, CosteOper, NombreOper) :-`

     `Especificación` (formada por precondiciones, acciones y por

     `\+peligro(Estado)` … evitar estados de peligro)
4. Coste del operador : representa el esfuerzo de aplicar dicho operador una vez
5. Función de coste de la solución *`suma coste operadores aplicados`*
6. Solución: camino desde el estado inicial a un estado objetivo
   - Pueden haber soluciones (caminos) de diferentes costes

→ ver documento : `convenioEspacioEstados.PDF`



## Convenio Representación JAVA: Problem.java

```
package aima.search.framework;
/**
• Tiene cuatro componentes:
• 1) Estado inicial.
• 2) Función sucesor.
• 3) Test de Objetivo.
• 4) Coste del camino.
*/
public class Problem {
   protected Object initialState;
   protected SuccessorFunction successorFunction;
   protected GoalTest goalTest;
   protected StepCostFunction stepCostFunction;
   protected HeuristicFunction heuristicFunction;
   protected Problem() {
                    }
```

## Convenio Representación JAVA: Implementar clases (1)

(1) Estado del Problema, define los elementos que participan:

Ej1: ej8p\src\problemas\eightpuzzle\EightPuzzleBoard.java

- Clase independiente, no es del Framework
  - **`public class EightPuzzleBoard`**
  - Definir la representación de cada estado: un tablero ("board"), un mapa, etc…
- Su constructor sirve para generar estados
  - **`public EightPuzzleBoard()`**
  - En particular se tiene que usar para generar el estado inicial
- Incluye los operadores para transformar un estado en otro estado válido
  - **`public void moveGapRight()`**
  - Son las acciones de los movimientos
- Incluye las precondiciones para aplicar un movimiento
  - **`public boolean canMoveGap(String where)`**
- Particular de JAVA, redefinir en el estado ("board")
  - **`hashCode()`** –ver más adelante–
  - **`equals()`**



## Convenio Representación JAVA: Implementar clases (2 y 3)

(2) Realiza los movimientos

- Obtener los estados sucesores (accesibles) del actual
- aima.search.framework.SuccessorFunction
- EJ1: EightPuzzleSuccessorFunction.java, …que tiene:

```
public class EightPuzzleSuccessorFunction
                                    implements SuccessorFunction
```

- Un movimiento (obtiene un sucesor)
  - Precondiciones: **`canMoveGap(String where)`**
  - Acciones: **`moveGapRight()`**

(3) Test para comprobar si el estado actual es el estado objetivo (definirlo dentro)

- aima.search.framework.GoalTest
- EJ1: EightPuzzleGoalTest.java, …que tiene:

```
public class EightPuzzleGoalTest
                                    implements GoalTest
```

# Ejecutar una Demo de Búsqueda: uso de la clase Problem

EJ1: \ejemplo\ej8p\src\problemas\demos\EightPuzzleDemo.java

El main(…) llama a este método:

```
package aima.search.framework;
   private static void eightPuzzleDLSDemo() {
       System.out.println("\nEightPuzzleDemo recursive DLS -->");
       try {
             Problem problem = new Problem(random1,←estadoinicial
                         new EightPuzzleSuccessorFunction(),
                         new EightPuzzleGoalTest());
             Search search = new DepthLimitedSearch(9);
             SearchAgent agent = new SearchAgent(problem, search);
             printActions(agent.getActions());
             printInstrumentation(agent.getInstrumentation());
       } catch (Exception e) {
             e.printStackTrace();
             }     }}
```



# Estado ("board") : hashCode() , equals() para tablas hash

- Tablas Hash : Almacenan colecciones de pares clave/valor
  - Usadas para las estructuras de "abiertos" y "cerrados" que almacenan los estados
- Cualquier objeto distinto de null puede ser tanto clave como valor
- La clase de las claves debe implementar los métodos hashCode() y equals() para poder hacer búsquedas y comparaciones
  - hashCode() => entero positivo único y distinto para cada clave (no varía durante la ejecución del programa)
  - dos claves iguales según equals() => hashCode() devuelve el mismo entero
- hashCode de Object
  - Genera una dirección de memoria
  - Dos objetos con igual contenido no tienen el mismo código hash, al estar en direcciones de memoria distintas
- Implementar hashCode del estado "EightPuzzleBoard"

```
public int hashCode() {
      int result = 17;
      for (int i = 0; i < 8; i++) { //cada estado:posiciones dígitos únicas
            int position = this.getPositionOf(i);
            result = 37 * result + position;
      }
      return result;    }     // suma de productos por números primos
```