**TPE-1  
Sistemas de Inteligencia Artificial**

E2 - Eternity Problem

Alan Pomerantz

Tomas Lori

Santiago Ramirez Ayuso

I.T.B.A 2015 - G.2

**Introducción al problema**

**Eternity II**

Eternity II, lanzado el 28 de Julio de 2007, es un puzzle plano que procede al Eternity I, de 256 piezas cuadradas que encajan en un tablero cuadrado de 16×16 casillas. Se requiere que las piezas coincidan en los colores adyacentes a otras. Fue diseñado con el objetivo de que por fuerza bruta el número de combinaciones requeridas para obtener su única solución sea computablemente imposible de lograr por su gran requerimiento de análisis combinatorio. (Se estiman 1.15 × 10661 posibilidades). Las piezas pueden usarse en sus 4 orientaciones, y se poseen 22 colores, además de los colores grises que representan paredes. 5 de esos colores solo pueden ser encontrados en los bordes y esquinas, y solo 17 pueden ser encontrados como piezas internas a los bordes.

Se otorgaba un premio de $2 millones de dólares a quien pudiera hallar su solución.  
 Actualmente, no se ha encontrado solución alguna a dicho problema.  
  
  
***Fig1. Eternity II***

**Enunciado**

Implementación de las estrategias de búsqueda no informadas: depth ﬁrst, breadth ﬁrst y profundización iterativa. Implementación de las estrategias de búsqueda informadas: greedy search y A\*. Heurísticas. Presentación de al menos dos (2) heurísticas. Las heurísticas deben ser no triviales. Enumerar las diferencias entre ellas. Todas las heurísticas presentadas deben estar codiﬁcadas y se debe poder ejecutar la búsqueda utilizando cualquiera de ellas. Especiﬁcar si son o no admisibles. Función de costo. Presentación de la/las funciones de costo creadas. Si hay más de una, enumerar las diferencias entre ellas. Todas las funciones de costo presentadas deben estar codiﬁcadas y se debe poder ejecutar la búsqueda utilizando cualquiera de ellas. Tanto las estrategias de búsqueda como las funciones heurísticas deberán ser entrada del programa a ﬁn de poder intercambiarlas sin necesidad de recompilar el código fuente. De cada corrida se deberá analizar al menos: la profundidad de la solución, cantidad total de estados generados, número de nodos frontera, número de nodos expandidos y tiempo de procesamiento.

**Primeros Pasos**

A la hora de preparar el desarrollo del sistema en su totalidad, la primera tarea era poder lograr entender la dinámica del juego y tratar de sacar conclusiones rápidas que brindaran un approach de cómo encarar el problema. Buscamos el juego disponible en internet para jugar, y realizamos fallidos intentos de lograr armar el tablero, que aunque no era muy grande (4x4) nos llevo más de 60 minutos poder hallar una solución. Frente a la lectura de diversos papers de investigación y tesis realizadas sobre la resolución del juego, sacamos la conclusión de que deberíamos relajar el problema a la confección de un resolutor de Eternity II de tamaño menor al objetivo inicial de 16x16.

Nuestra propuesta de implementación, basada en la relajación del problema inicial, propone la resolución de tableros de hasta 7x7 de dimensión. Y hace uso de dos distintas heurísticas aplicadas para los métodos de búsqueda informadas, distinguidas en su forma de ubicar piezas dentro del tablero. Las mismas serán analizadas en detenimiento en el presente informe, y surgen de haber intuido posibles formas de llenar el tablero con piezas, respetando la posición de los bordes.

**Definición del problema y modelización**

**Estado inicial**: Tablero vacío y lista de fichas a ubicar con tantas fichas como dimension tenga el tablero.

**Conjunto de posibles acciones**: dado un estado, se puede colocar sobre el tablero cualquier ficha siempre y cuando esta no haya sido colocada antes y no haya una ficha en la posición que se quiere colocar.

**Modelo de transición:** Al aplicar una regla sobre el tablero, se retorna un tablero nuevo con la pieza que se agregó en la posición indicada.

**Condición de Terminación**: dado que las fichas solo se insertan si el tablero que generan sigue siendo válido, nuestra condición de corte es que no haya mas fichas en la lista de fichas a ubicar. Esto solo ocurrirá cuando el tablero este completo y bien formado.

***Generamos las diversas clases necesarias para poder modelizar el problema.***

**Tile**   
Se cuenta con la clase Tile que representa una ficha del tablero, de manera optimizada, a través de sus 4 caras. Inicialmente se distinguió las 4 posibilidades con un numero entero indicando si era la cara up, down, left o right, pero luego, rápidamente se lo optimizo en un único valor entero que representa un patrón o pattern de bits de dirección, y que cuenta con las operaciones propias para poder rotar la ficha de manera rápida y en una única operación.   
  
**Board**La clase board representa a un tablero mediante una matriz de piezas, que posee una dimensión dada y una matriz en donde se ubican fichas, y cuyo espacio vacío se representa por null. También el tablero posee la posibilidad de ser rotado, ya que se ha implementado un método que otorga dicha posibilidad. También es el tablero quien cuenta con el método que inserta una ficha en él mismo, validando los límites de su contorno.

**GPS Package**

Se utiliza el paquete GPS provisto por la cátedra, intentando mantenerlo lo más original posible. La única modificación realizada, es que se le agrego a los nodos (GPSNode) un entero, contador de profundidad, necesario para la profundización iterativa.

**Reglas**

Codificamos una meta regla que consolida la acción de insertar una ficha en una coordenada dada por un X y un Y. La regla es básicamente insertar una ficha en el tablero si el tablero es un tablero valido no pudiendo inserter fichas con un lado con patron pared en las esquinas o el relleno así como tampoco un relleno en algun contorno o una ficha con 2 lados con patron pared en algun lugar que no sea esquina.

**Métricas y Función de Costo**

Como este problema tiene solucion en una cantidad fija de pasos decidimos tomar como metrica la facilidad de ubicar las fichas en el tablero y no los pasos restantes para completarlo. Se consideraron 3 subconjuntos de fichas con distinta facilidad de ubicacion, planteando las esquinas como el subconjunto mas simple de ubicar ya que siempre seran 4 fichas, seguido por los bordes y finalmente por el relleno.   
Dicho esto, se desarrolló una única función de costo tomando en cuenta la dificultad de ubicar una pieza siendo las esquinas las de menor costo, seguidas por los bordes y finalmente por el relleno el cual posee el mayor costo.

**Heurísticas**

**Heurística 1: Peso de la ficha con recorrido de arriba hacia abajo por filas**La primera heurística corresponde con llenar el tablero de manera lineal, comenzando por la esquina superior izquierda, haciendo un barrido horizontal insertando fichas.   
Para lograr esto le dimos un valor muy alto a los tableros que no tengan sus fichas consecutivas en el orden previamente mencionado de forma tal que estas no se exploren mientras que al resto de los estados le otorgamos un valor en base a la dificultad de ubicacion de las fichas que nos restan colocar. De esta manera un tablero mas completo da un valor heurístico menor ya que esta mas cerca de ser completado y por lo tanto la cantidad de fichas restantes es menor.

**Heurística 2: Peso de la ficha con recorrido en espiral**La segunda heurística corresponde con llenar el tablero de manera circular, comenzando por la esquina superior izquierda, recorriendo todo el contorno primero y luego realizando la misma accion reduciendo la dimension del espiral hasta llegar al centro.

***Ambas heurísticas presentadas son no admisibles. En el caso en que una pieza que no corresponda sea ubicada en una posición en la que no sea posible alcanzar una solución (la única), la función de evaluación obtendrá un valor heurístico que superará al valor de la suma de los pesos de las NxN fichas totales, que corresponden al costo de lograr la única solución posible del juego***

**Conclusiones**Se concluye la diferenciación entre rendimiento entre algoritmos informados y no informados. Dado que las reglas aplicadas pueden dejar al estado como un nodo que no llegara a la solución, la cantidad de nodos sin aplicar heurística crece abruptamente. Los resultados se pueden observar en la sección Anexo. La heurística que ubica por espirales frente a la primer heurística, la que ubica fila por fila, es relativamente más eficiente, ya que al poner todas las paredes primero se reduce más el problema (la ubicación de las fichas comprometidas de los bordes y esquinas) que ubicando la matriz por filas.   
  
Probablemente la combinación de una heurística donde se ubique inicialmente todos los bordes, y luego las fichas de relleno acorde con los mismos (una vez colocadas absolutamente todas las paredes) ayudaría también a encontrar una solución más rápidamente usando, por qué no, como función de costo y heurística posible, las distancias de Manhattan al centro del tablero.   
Encontramos también las diferencias de rendimiento entre utilizar Greedy vs A\*. Siendo el primero más performante, encontrando más rápido la única solución posible del tablero dado

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Anexo.** | **Dimensión 2x2** | | | | |
|  | Tiempo(s) | Profundidad | Nodos Expandidos | Nodos Frontera | Estados Generados |
| **BFS** | 14.84 seg. | 4 | 12225 | 15359 | 27585 |
| **DFS** | 0.02 seg. | 4 | 4 | 96 | 101 |
| **IDFS** | 0.04 seg. | 4 | 14 | 94 | 99 |
| **Greedy (H1)** | 0.02 seg. | 4 | 4 | 94 | 99 |
| **Greedy(H2)** | 0.02 seg. | 4 | 4 | 94 | 99 |
| **Astar(H1)** | 0.03 seg. | 4 | 4 | 94 | 99 |
| **Astar(H2)** | 0.03 seg. | 4 | 4 | 94 | 99 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | **Dimensión 4x4** | | | | |
|  | Tiempo(s) | Profundidad | Nodos Expandidos | Nodos Frontera | Estados Generados |
| **BFS** | \* | \* | \* | \* | \* |
| **DFS** | \* | \* | \* | \* | \* |
| **IDFS** | \* | \* | \* | \* | \* |
| **Greedy (H1)** | 1.07 seg. | 16 | 30 | 5694 | 5725 |
| **Greedy(H2)** | 0.8 seg. | 16 | 18 | 4600 | 4619 |
| **Astar(H1)** | 114.39 seg. | 16 | 27 | 4980 | 5008 |
| **Astar(H2)** | 109.92 seg. | 16 | 27 | 4953 | 4992 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  | **Dimensión 5x5** | | | | |
|  | Tiempo(s) | Profundidad | Nodos Expandidos | Nodos Frontera | Estados Generados |
| **BFS** | \* | \* | \* | \* | \* |
| **DFS** | \* | \* | \* | \* | \* |
| **IDFS** | \* | \* | \* | \* | \* |
| **Greedy (H1)** | 2041.91 seg. | 25 | 403 | 231889 | 232293 |
| **Greedy(H2)** | 12137.44 seg. | 25 | 323 | 168296 | 168620 |
| **\*\*\* Astar(H1)** | 0.45 seg. | 25 | 65 | 12 | 78 |
| **\*\*\* Astar(H2)** | 0.77 seg. | 25 | 132 | 14 | 147 |
|  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Referencias:** \* Cortó por tiempo (tiempo mayor a 1 hora) | |  |  |  |
| \*\* Cortó por memoria |  |  |  |  |
| \*\*\* Poda de estados para optimización (sin aplicar esta poda el algoritmo no termina por tiempo) | | | | |