Universidad Zaragoza
1542

Trabajo Fin de Grado

Aprendizaje de IA basado en el pensamiento
Thinking-based AI learning

Autor

Ignacio Gutiérrez Villar

Directora

Piedad Garrido Picazo

Escuela Universitaria Politécnica de Teruel

2022

Agradecimientos

Quiero agradecer a todos los profesores que he tenido durante estos años de carrera, por enseñarme los conocimientos necesarios para poder realizar mi TFG.

También a mi tutora, Piedad Garrido, por la ayuda prestada durante el TFG y todas las explicaciones que me ha dado durante este periodo.

Además, quisiera mencionar a mi compañero Sergio, a mi amigo Rubén y a mi pareja Eva por apoyarme en este último tramo de la carrera.

Por último, quisiera agradecer a mi familia por animarme durante todos estos años de enseñanza.

Resumen

El objetivo de este Trabajo Final de Grado (TFG) es hacer más comprensivo el enfoque basado en agentes (percepción ---> procesamiento de información ----> acción), de la asignatura de Inteligencia Artificial (IA) del Grado en Ingeniería Informática (GII), a través del Aprendizaje Basado en el Pensamiento (Thinking-based Learning).
Tanto las técnicas de búsqueda como el aprendizaje automático de IA se utilizan en la resolución de problemas, con o sin incertidumbre, que requieren una profunda comprensión de los conceptos, colocando al alumno en el centro de su propio aprendizaje.

Al tratarse de un procedimiento largo y complejo, si se quiere enfocar el aprendizaje del alumnado sólo al funcionamiento de las técnicas y/o del aprendizaje automático, intentando mejorar su capacidad analítica, su pensamiento crítico y creativo e incluso su inteligencia emocional, se aborda el problema planteando la resolución y automatización de juegos de mesa clásicos o la simulación de juegos analógicos y/o digitales. Este TFG usará el juego del Sudoku para el aprendizaje de los conocimientos de la asignatura, basándose en la enseñanza de métodos de búsqueda para la resolución de un CSP (Constraint satisfaction problem).

Palabras Clave

Sudoku, CSP (Constraint satisfaction problem), Búsqueda, IA (Inteligencia Artificial), Heurística.

Abstract

The objective of this Final Degree Project (TFG) is to make the agent-based approach (perception ---> information processing ----> action), of the Artificial Intelligence (AI) subject of the Informatic Engineering Degree(GII) more comprehensive., through Thinking-based Learning.
Both search techniques and AI machine learning are used in solving problems, with or without uncertainty, that require a deep understanding of concepts, placing the student at the center of their own learning.

As it is a long and complex procedure, if you want to focus the students' learning only on the functioning of the techniques and/or automatic learning, trying to improve their analytical capacity, their critical and creative thinking and even their emotional intelligence, the problem posing the resolution and automation of classic board games or the simulation of analog and/or digital games. This TFG will use the Sudoku game to learn the knowledge of the subject, based on the teaching of search methods to solve a CSP (Constraint satisfaction problem).

KeyWords

Sudoku, CSP (Constraint satisfaction problem), Search, AI (Artificial Intelligence), Heuristic.

Tabla de contenidos

1. Introducción y objetivos 1
2. Estado del Arte 4
3. Análisis, diseño, implementación y pruebas 8
    3.1. Análisis 8
    3.2. Diseño 14
    3.3. Implementación 16
    3.4. Pruebas y resultados 24
    3.5. Tecnologías utilizadas 25
4. Accesibilidad y Usabilidad 27
5. Licencia software y documental 31
6. Conclusiones y trabajo futuro 33
7. Referencias Bibliográficas 34
8. Anexos 36
    Enunciado de boletín de prácticas 36

1. Introducción y objetivos

Este Trabajo de Fin de Grado (TFG) consiste en la realización del análisis, diseño e implementación de un programa que resolverá sudokus planteando como un CSP el problema, y encontrando las soluciones mediante IA a través del aprendizaje basado en el pensamiento. La motivación que llevó a su realización fue la buena experiencia en la asignatura de IA del grado. Por ello se quería transmitir mediante la creación de una aplicación de uso académico el mismo aprendizaje satisfactorio a otras generaciones de estudiantes.

Para la realización del TFG se han planteado los siguientes objetivos:

- Resolver el problema mediante la exploración de árboles y grafos, con nodos estado.
- Utilizar para la resolución tanto métodos de búsqueda informada como no informada.
- Plantear un programa y un enunciado que sea de utilidad para el aprendizaje del alumnado de la asignatura de IA.

Además, se ha propuesto el Objetivo de Desarrollo Sostenible (ODS) de educación de calidad, en concreto 4.4. que consiste en aumentar considerablemente el número de jóvenes y adultos que tienen las competencias necesarias, en particular técnicas y profesionales, para acceder al empleo, el trabajo decente y el emprendimiento.[7]

El método de enseñanza promovido por este TFG va a ser el aprendizaje basado en el pensamiento, Thinking-Based Learning (TBL), que es una metodología activa que enseña a los alumnos a pensar, razonar, tomar decisiones y construir su propio aprendizaje a través del trabajo de los temas enseñados en las clases teóricas del Grado de Ingeniería Informática (GII).

Este problema se puede encontrar resuelto comúnmente mediante Backtracking, como se verá más adelante en el apartado de Estado del Arte, y sin un objetivo didáctico como el del presente documento. El Backtracking (o vuelta atrás) es una técnica algorítmica para encontrar soluciones a problemas que tienen una solución completa, en los que el orden de los elementos no importa, y en los que existen una serie de variables, a cada una de las cuales, debemos asignarle un valor teniendo en cuenta unas restricciones dadas.[4]

Respecto al sudoku es un juego matemático que se inventó a finales de la década de 1970, adquirió popularidad en Japón en la década de 1980, y se dio a conocer en el ámbito internacional en 2005, cuando numerosos periódicos empezaron a publicarlo en su sección de pasatiempos.[12]

El objetivo del sudoku es rellenar una cuadrícula de 9 × 9 celdas (81 casillas) dividida en subcuadrículas de 3 × 3 (también llamadas "cajas" o "regiones") con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas de las celdas. La forma inicial del juego es que sean nueve elementos diferenciados, que no se deben repetir en una misma fila, columna o subcuadrícula. Un sudoku bien planteado sólo puede tener una solución.[12]

En cuanto los Problemas de satisfacción de restricciones (CSP, por sus siglas en inglés), son problemas matemáticos definidos como un conjunto de objetos tal que su estado debe satisfacer un número de restricciones o limitaciones. Los CSP representan las entidades de un problema como una colección homogénea finita de restricciones sobre variables, las que son resueltas por métodos de satisfacción de restricciones. Los CSP son el tema de una intensa investigación en Inteligencia Artificial e Investigación de operaciones, dado que la generalidad en su formulación provee un principio básico para analizar y resolver problemas de distintos tipos. Los CSP a menudo muestran gran complejidad, requiriendo una combinación de métodos heurísticos y búsqueda combinatoria para ser resueltos en un tiempo razonable.[8]

El restante del presente documento va a seguir la siguiente estructura:

- **Estado del Arte:** Dónde se compararán proyectos de software libre y propietario con el presente TFG, y también las investigaciones que haya respecto a la temática de IA con el sudoku aplicada a la enseñanza
- **Análisis, diseño e implementación:** Se explicará que análisis se ha realizado y el diseño de la aplicación en sí, así como la formas en que ha sido implementada. También se comentarán las tecnologías utilizadas.
- **Accesibilidad y usabilidad:** En este apartado se realiza un análisis respecto a la accesibilidad y usabilidad de la aplicación.
- **Conclusiones y trabajo futuro:** El apartado final donde se comentarán las conclusiones sacadas de la realización del TFG y también las ideas de trabajo futuro.
- **Anexos:** Aquí se plantea un posible boletín de prácticas adaptado al alumnado de tercer curso, primer cuatrimestre del GII.

2. Estado del Arte

En este apartado se van a analizar diferentes proyectos de juegos desde el punto de vista de un CSP. Para ello se analizará si utilizan Inteligencia Artificial (IA) o no, los métodos de IA utilizados y las tecnologías con las que se desarrolla el proyecto en cuestión. Se van a analizar tanto proyectos de software libre como propietario. Además, se van a analizar los artículos de investigación relacionados con el tema.

Software libre:

Proyecto 1:

Este proyecto realizado en Python utiliza la propagación de restricciones empleando el algoritmo Arc Consistency #3 (AC-3), y el método de búsqueda depth-first search (DFS) con Backtracking usando las heurísticas de Minimum Remaining Value (MRV) heuristic y Forward Checking (FC).

Proyecto 2:

Este proyecto se plantea de cara a enseñar a otros usuarios el modelo "learn by doing", que presupone que el lector va programando, aprendiendo en Python y experimentando, a medida que va siguiendo el post del proyecto. Utiliza el método de backtracking para resolver los sudokus.

Proyecto 3:

Este proyecto es un solucionador de Sudoku implementado en Python basado en CSP. Usa el problema de satisfacción de restricciones (CSP), además de aplicar las heurísticas de degree y MRV. Si hay una solución para el problema del Sudoku el programa informará de ello. De lo contrario, la búsqueda no tiene éxito y no se obtiene una solución.

Proyecto 4:

Es un proyecto desarrollado en Python que usa el algoritmo de Backtracking con dos heurísticas diferentes (MRV y degree), una implementación del algoritmo de búsqueda min-conflicts para resolver Sudoku, y también una aplicación por consola de Python simple que lee un sudoku de input.py y lo resuelve usando uno de los algoritmos anteriores.

Proyecto 5:

Es un proyecto realizado en Java que lee cualquier Sudoku de un archivo y le encuentra una solución. Utiliza el algoritmo Backtracking recursivo. También muestra cuánto tiempo lleva encontrar la solución a un sudoku en particular.

Software propietario:

Proyecto 6:

Este software ofrece una solución a los usuarios que quieren resolver rompecabezas de Sudoku ingresando una cuadrícula parcialmente completa. El usuario ingresa los números a la izquierda y un clic en el botón mostrará la solución a la derecha. Tanto los puzles como sus soluciones se pueden imprimir.

Proyecto 7:

Este software desarrollado en C + + ofrece solucionar un Sudoku 9x9, poniendo el usuario el sudoku inicial. Contiene varios métodos lógicos de resolución, así como selección de fuerza bruta.

N.º Proyecto	Denominación	URL
1	Sudoku_py	https://aurbano.github.io/sudoku_py/
2	Sudoku	https://github.com/jorditorresBCN/Sudoku
3	Sudoku_solver	https://github.com/erfanghasemi/sudoku_solver?ref=https://githubhelp.com
4	SudokuCSF	https://github.com/mahdavipanah/SudokuPyCSF
5	Sudoku-Solver	https://github.com/devkapupara/Sudoku-Solver
6	Sudoku Solver Software	https://es.freedownloadmanager.org/Windows-PC/Sudoku-Solver-Software.html
7	Sudoku Solver	https://www.linux-apps.com/p/1407271

Tabla 1. Nombre y URL de los proyectos.

N.º Proyecto	Tecnologías	Software	Tipo de búsqueda
1	Python	Libre	Informada
2	Python	Libre	No informada
3	Python	Libre	Informada
4	Python	Libre	Informada
5	Java	Libre	No informada
6	Java	Propietario	No informada
7	C++	Propietario	No informada

Tabla 2. Tecnologías, tipo de software y búsquedas usadas por los proyectos.

En cuanto a los proyectos analizados, no hay ninguno que plantee un acercamiento al problema mediante ambos tipos de búsqueda, tanto no informada como informada. Además, este TFG tiene un objetivo didáctico añadido de enseñar, que no tienen los analizados, mediante el uso de éste ambos tipos de búsquedas y diferentes formas dentro de estos tipos.

Ahora, en cuanto artículos de investigación relacionados con el uso de IA en la resolución de sudokus se han seleccionado los siguientes:

N.º Articulo	Autores	Universidad
1	M.Eremic, R. Adamov, N. Bilinac, V. Ognjenovic, V. Brtka, I. Berkovic	University of Novi Sad, Technical faculty “Mihajlo Pupin”, Zrenjanin, Republic of Serbia
2	Tirsa Ninia Lina, Matheus Supriyanto Rumetna	Faculty of Computer Science, Victory University of Sorong, Indonesia
3	Yuji Sato, Hazuki Inoue	Faculty of Computer and Information Sciences, Hosei University, Tokyo, Japan
4	Tristan Cazenave	Dept. Informatique Universite Paris ,Saint-Denis, France

Tabla 3. Autores y centros universitarios de los artículos de investigación.

Artículo 1:

Este trabajo trata de las diferencias entre dos algoritmos utilizados para resolver problemas de Sudoku. Los algoritmos que se utilizan son el algoritmo de fuerza bruta y el Backtracking. El proyecto está hecho en Java. El objetivo de este proyecto es mostrar las diferencias entre estos dos algoritmos. También hay algunos ejemplos de algoritmos clásicos para la resolución de problemas que se utilizaron junto con Backtracking y Brute Force.

Artículo 2:

El propósito de esta investigación es encontrar una solución para Sudoku y hacer un análisis comparativo de los resultados de búsqueda de los dos algoritmos: Breadth First Search (BFS) y Depth Limited Search (DLS).

Artículo 3:

En este artículo se proponen operaciones genéticas que consideran bloques de construcción efectivos para usar algoritmos genéticos para resolver rompecabezas de Sudoku. También se propone una función de búsqueda local más potente. Evaluación de las técnicas propuestas utilizando conjuntos comerciales de Sudoku y tres rompecabezas clasificados como super difíciles.

Artículo 4:

En este artículo se detallan los algoritmos de búsqueda utilizados para resolver problemas de Sudoku con satisfacción de restricciones. Se muestra que una buena heurística ayuda a resolver problemas de manera más eficiente.

Respecto a los artículos analizados se puede observar que la investigación está centrada en la resolución en sí del sudoku y la eficiencia de ésta. Este TFG se centra además en la utilización de estos métodos, aplicados en el sudoku, para una labor educativa en las aulas universitarias.

N.º Proyecto	Algoritmos
1	Backtracking y Brute Force.
2	Breadth First Search y Depth Limited Search
3	Algoritmos genéticos
4	Forward Checking y Limited Discrepancy Search

Tabla 4. Algoritmos usados en los proyectos de los artículos.

3. Análisis, diseño, implementación y pruebas

Este apartado va a seguir la disciplina de la ingeniería de software que está formada por un conjunto de métodos, herramientas y técnicas que se utilizan en el desarrollo de los programas informáticos.[3]

Esta disciplina trasciende la actividad de programación, que es el pilar fundamental a la hora de crear una aplicación.

La metodología por realizar consiste en el análisis previo de la situación, el diseño del proyecto, el desarrollo del software, las pruebas necesarias para confirmar su correcto funcionamiento y la implementación del sistema.[11]

3.1. Análisis

En este apartado se va a realizar un análisis de cómo afrontar el problema del sudoku planteando como un problema de nodos estado.

- Sudoku inicial
    - Se tiene al menos 17 números colocados en las celdas.
- Definir Objetivo
    - Rellenar todas las celdas con números
- Definir problema
    - Estados: diferentes números posibles en celdas
    - Acciones: probar un número que cumpla las normas del sudoku
- Solución
    - Secuencia de números en celdas que completen un sudoku que cumpla las normas.

Las estructuras de datos que se recorren por los métodos de búsqueda, que se explicaran a continuación, son el árbol y el grafo. Ambos son estructuras no lineales, es decir representaciones gráficas, en las que se conectan un conjunto de nodos que expresan las relaciones que se dan entre ellos En el árbol se expande por hojas con orden jerárquico y en el grafo no. Además, en el grafo no se pueden repetir dos nodos iguales.[1]

Métodos de búsqueda

Para encontrar la solución nombrada, se van a plantear los siguientes métodos de búsqueda [10]:

- **Búsqueda no informada (ciega):**

  Las estrategias no disponen de información adicional sobre estados más allá de la proporcionada en la definición del problema, por lo que sólo pueden expandir estados y distinguir entre estados objetivos y no-objetivos. Dentro de esta categoría se distinguen:

  - **Búsqueda primero en anchura (breadth-first):**

    Se expanden primero los estados de un nivel antes de expandir los del estado siguiente.

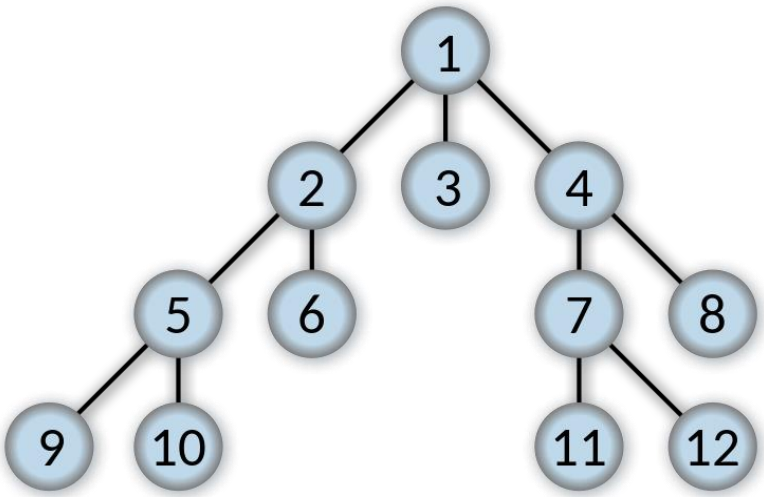


Figura 1. Búsqueda primero en anchura.

- **Búsqueda primero en profundidad (depth-first):**

  Se expanden primero los estados de mayor nivel (los más profundos).

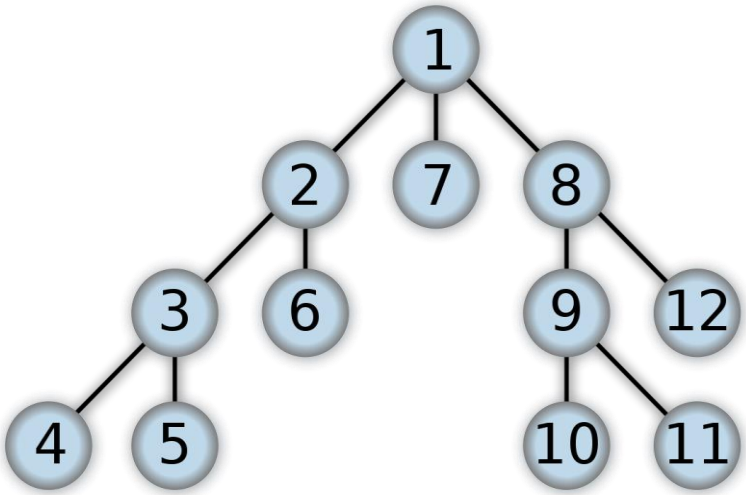


Figura 2. Búsqueda primero en profundidad.

- **Búsqueda en coste uniforme (uniform cost):**

  Se expanden primero los estados con menor coste acumulado de acciones para llegar a él (función g).

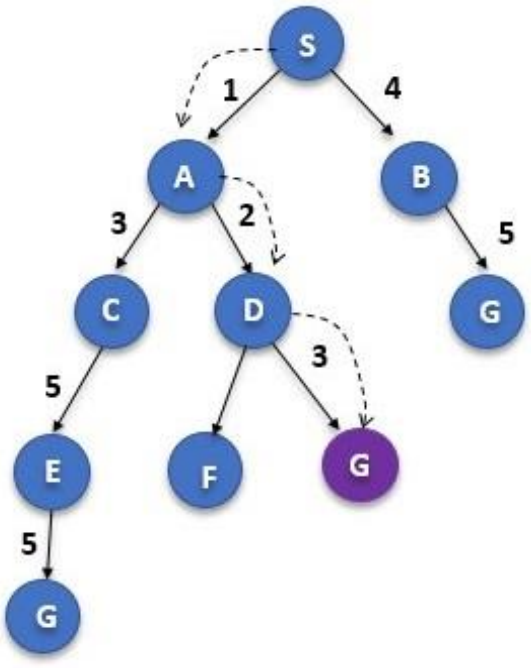


Figura 3. Búsqueda en coste uniforme

- **Búsqueda informada (heurística):**

  Se utiliza información propia del problema más allá de la proporcionada en la definición. Se utiliza una función heurística, normalmente denominada h(n), que estima el coste del camino más corto desde el estado actual hasta el objetivo. Para que la heurística sea admisible, nunca puede indicar un coste mayor que el real para algún estado. En esta categoría se incluirían:

  - **Búsqueda voraz, avariciosa o primero el mejor (greedy or best-first):**

    Se expanden primero los estados con el menor valor de la función heurística h.

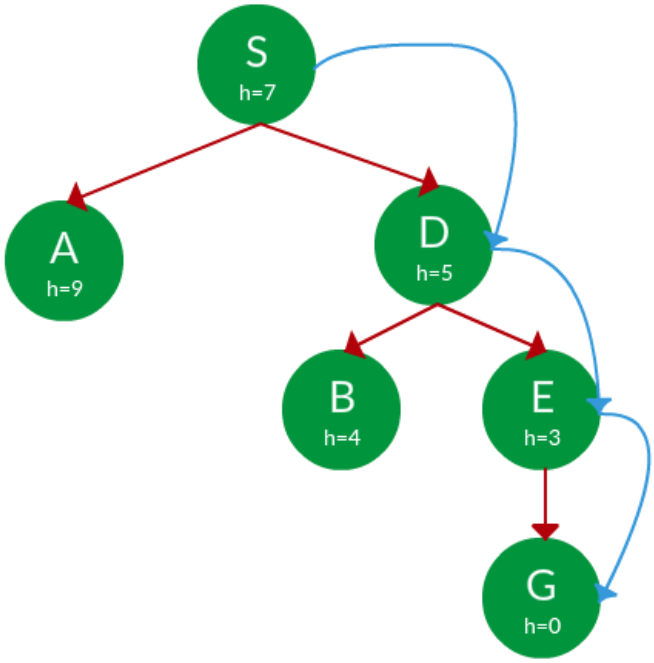


Figura 4. Búsqueda voraz.

- **Búsqueda A*:**

  Se expanden primero los estados con menor función f, obtenida como el valor de la suma del coste para llegar al estado más la estimación de alcanzar el estado objetivo, es decir, f = g + h.

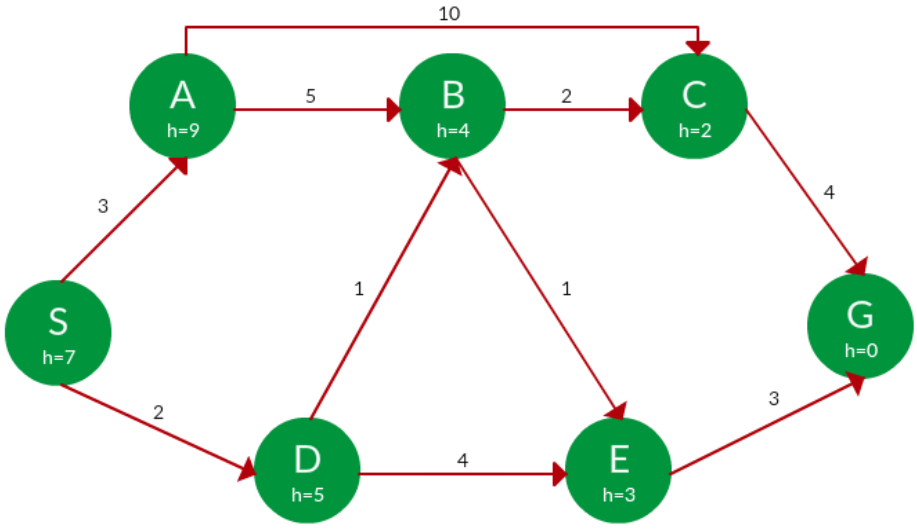


Figura 5. Búsqueda A*.

Heurísticas

La heurística, como disciplina científica, y en su sentido amplio puede ser aplicada a cualquier ciencia con la finalidad de elaborar medios, principios, reglas, estrategias como ayuda para lograr encontrar la solución más eficaz y eficiente al problema que analiza el individuo.

En el ámbito de la Inteligencia artificial se usa el término heurístico para describir una clase de algoritmos que aplicando el conocimiento propio del problema y técnicas realizables se acercan a la solución de problemas en un tiempo razonable.[2]

Respecto a las heurísticas que serán usadas en las técnicas de búsqueda informada en este TFG son las siguientes [5]:

- **Heurística 1:**

  Heurística basada en el número de huecos vacíos restantes. Da prioridad a los estados más avanzados en el problema con menos huecos vacíos.

- **Heurística 2:**

  Heurística basada en el número de nodos expandibles que tiene cada nodo. Se calculan los nodos posibles a partir del nodo objetivo y se da prioridad a los nodos con menos nodos por expandir.

3.2. Diseño

En la fase de diseño es dónde se va a especificar qué clases se van a tener que implementar, cómo van a estar relacionadas entre sí, cómo se van a almacenar todos los datos que se usen en la aplicación y cuál va a ser el proceso que va a seguir la aplicación a la hora de realizar una función.

Diagrama de clases

Aquí se va a mostrar las entidades que va a utilizar la aplicación, los atributos y métodos que contiene cada una de ellas, así como las relaciones que existen.
A continuación, se pasa a mostrar el diagrama de clases [9]:

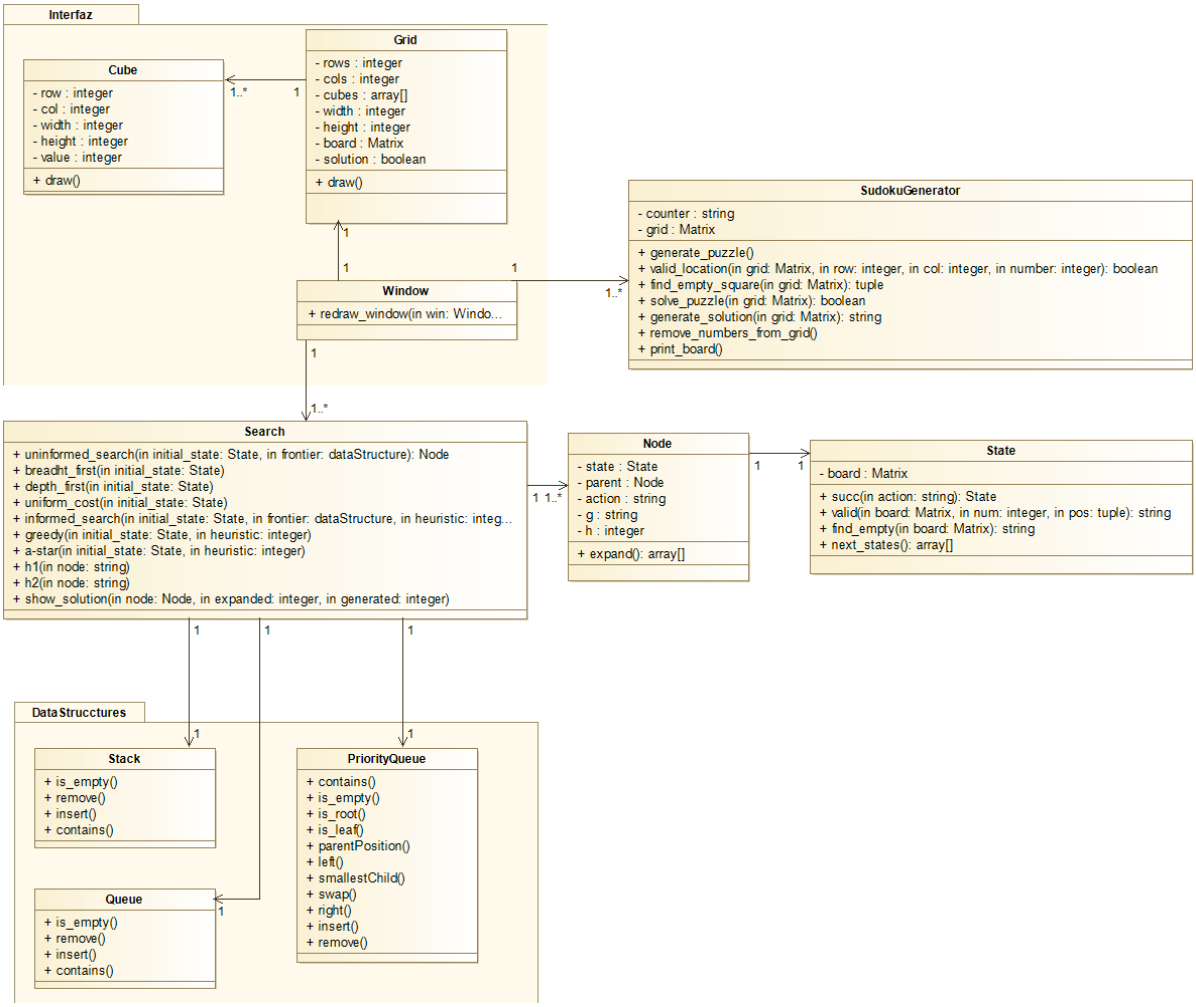


Figura 6. Diagrama de clases.

La interfaz de este TFG se ha decidido que sea muy simple dado sus necesidades académicas de ser de fácil uso y claridad máxima. Por ello su diseño consiste en ser una única ventana que contiene una cuadrícula de sudoku, que se irá actualizando, y dos botones para realizar las únicas acciones que interesa al usuario: crear sudokus y resolverlos.

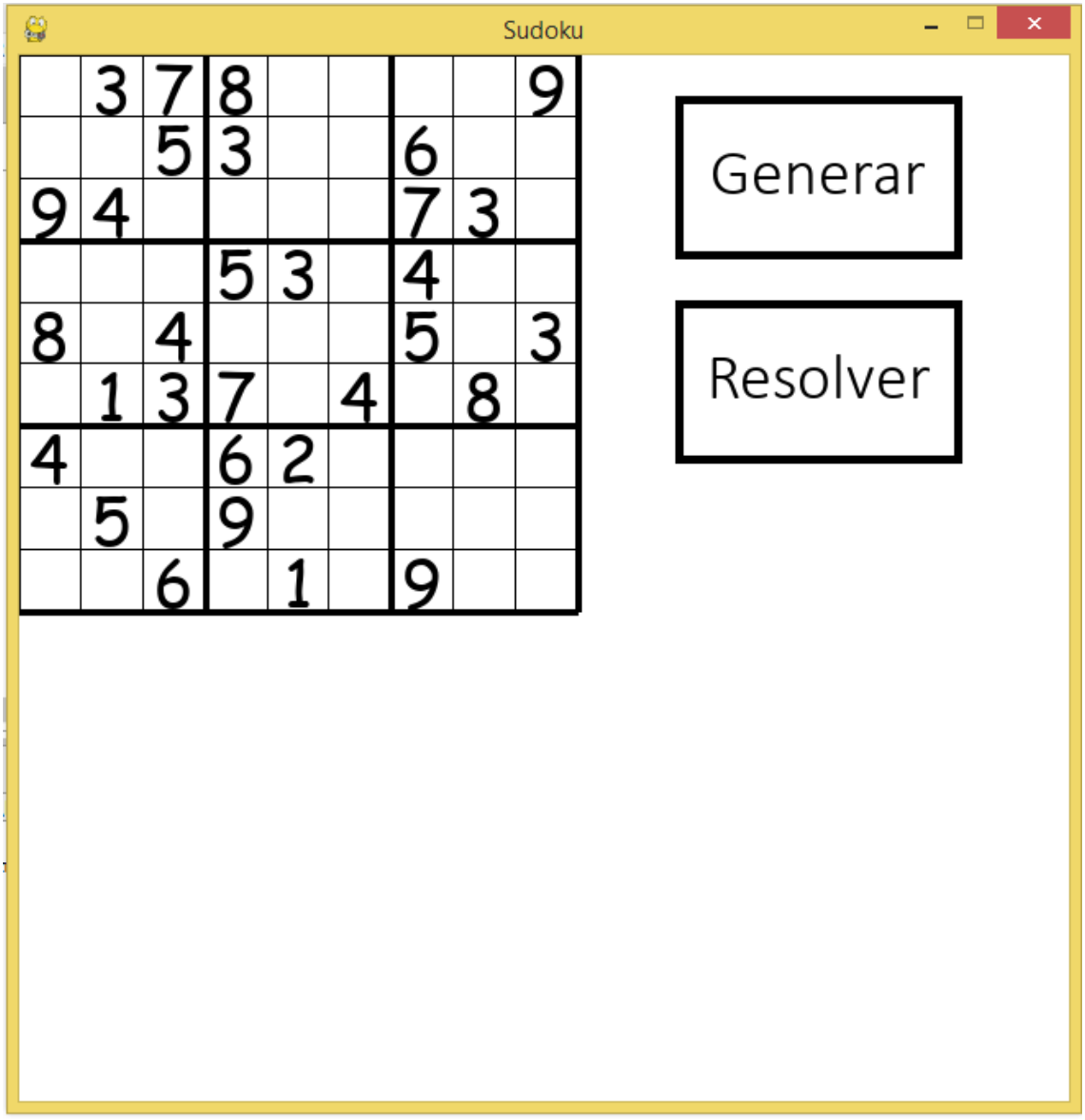


Figura 7. Interfaz de Usuario.

3.3. Implementación

En este apartado se van a explicar cómo se han implementado al final las clases diseñadas en el diagrama de clases, y cómo funcionan sus métodos concretamente.[14][15] Todo el código generado en este TFG esa disponible en github.com/696886/SudokuGame.

Clase SudokuGenerator

Esta clase sirve para generar estados iniciales del sudoku. En ella se genera una matriz vacía, después se añaden números aleatorios y se resuelve el sudoku sobre ello. Una vez comprobado que tiene 1 única solución se remueven números aleatoriamente dejando al menos 17 números iniciales.
A continuación, se van a explicar los métodos de la clase más detalladamente.

Primero el sudokuGenerator utiliza generate_puzzle que es un método que llama a otros dos, generate_solution y remove_numbers_from_grid.

```
def generate_puzzle(self):
    """Genera un nuevo sudoku ,lo resuelve y le quita numeros"""
    self.generate_solution(self.grid)
    self.remove_numbers_from_grid()

    return
```

Figura 8. Método generate_puzzle.

En generate_solution lo que se realiza es generar una solución de sudoku desde cero mediante backtraking. Se ponen números aleatorios en posiciones aleatorias, pero cumpliendo las normas del juego.

```
def generate_solution(self, grid):
    """Genera una solución con backtracking"""
    number_list = [1,2,3,4,5,6,7,8,9]
    for i in range(0,81):
        row=i//9
        col=i%9
        if grid[row][col]==0:
            random.shuffle(number_list)
            for number in number_list:
                if self.valid_location(grid,row,col,number):
                    grid[row][col]=number
                    if not self.find_empty_square(grid):
                        return True
                    else:
                        if self.generate_solution(grid):
                            return True
            break
    grid[row][col]=0
    return False
```

Figura 9. Método generate_solution.

Una vez realizado esto se utiliza remove_numbers_from_grid, donde se va a generar un tablero inicial quitándole números a la solución generada. Para que sea una solución válida, como definen los estándares del juego del sudoku, se comprueba también que el tablero inicial tenga al menos 17 pistas es decir 17 números iniciales dados al usuario. También se comprueba mediante Backtracking en diferentes iteraciones que el tablero resultante solo tenga 1 solución válida como debe ser en un sudoku bien formado.

```
def remove_numbers_from_grid(self):
    """
    Quita numeros de la matriz aleatoriamente comprobando que se queden
    y que no tenga mas de una solución el sudoku
    """
    non_empty_squares = self.get_non_empty_squares(self.grid)
    non_empty_squares_count = len(non_empty_squares)
    rounds = 3
    while rounds > 0 and non_empty_squares_count >= 17:
        #Al menos 17 numeros iniciales
        row,col = non_empty_squares.pop()
        non_empty_squares_count -= 1
        #Numero borrado que se puede necesitar poner otra vez
        removed_square = self.grid[row][col]
        self.grid[row][col]=0
        #Copia del problema a resolver
        grid_copy = copy.deepcopy(self.grid)
        #Contador de soluciones
        self.counter=0
        self.solve_puzzle(grid_copy)
        #si hay mas de una solucion se pone el ultimo numero borrado
        if self.counter!=1:
            self.grid[row][col]=removed_square
            non_empty_squares_count += 1
            rounds -=1
    return
```

Figura 10. Método remove_numbers_from_grid.

Clase Node

Esta clase representa los nodos de un grafo. Cada nodo contiene una representación del estado del problema, una referencia al nodo padre, un string que describe la acción que generó el nodo desde el nodo padre y el coste de camino g desde el origen.

Cada nodo puede expandirse mediante el método expand(), en él se utiliza el estado del nodo y se miran los siguientes estados posibles. Con estos estados nuevos posibles y las acciones que llevarían a ellos se crean nodos sucesores que se devuelven en un array de nodos. Estos se usarán en las búsquedas.

```
def expand(self):
    successors = []
    for (newState, action) in self.state.next_states():
        newNode = Node(newState, self, action)
        successors.append(newNode)
    return successors
```

Figura 11. Método expand.

Clase Search

En cuanto a la búsqueda hay dos tipos: no informada e informada. La implementación de ambas es parecida excepto que en la informada se tiene en cuenta la heurística utilizada para la frontera.

Respecto a la no informada se tiene como parámetros iniciales un estado inicial y una frontera. La frontera es la estructura de datos utilizada para organizar los nodos que se van a recorrer, cada estructura se utilizará para realizar un tipo de búsqueda diferente según el comportamiento de estas.

Durante la búsqueda se irán incrementando dos contadores de nodos expandidos y generados. La búsqueda se realiza en un bucle que irá realizando lo siguiente, primero comprobará si quedan posibles estados en la frontera. Si no quedan quiere decir que ese método de búsqueda no ha encontrado solución. Si hay un nodo en la frontera, se comprueba primero que no haya ningún hueco vacío en la matriz del estado del último nodo lo cual querría decir que se ha encontrado la solución.

En caso negativo respecto a encontrar la solución se saca de la frontera un nodo y se añade como explorado. Después se expande y se guardan sus nodos expandidos en un array. A continuación, se comprueba que los nuevos expandidos no se encuentren ya en explorados ni en la frontera y si cumplen esas dos condiciones se añaden a la frontera como posibles nodos a explorar.

En caso de que durante el bucle se encuentre solución se devolverá el último nodo explorado junto al número de nodos generados y expandidos durante el proceso.

Respecto a la búsqueda informada sería el mismo procedimiento, pero teniendo en cuenta las heurísticas para darle un valor a cada opción de nodo dentro de la estructura de datos.

```
def uninformed_search(initial_state, frontier):

    """
    Parametros:
       initial_state: estado inicial de busqueda (objeto de clase State)
       frontier: estructura de datos para contener los estados de la frontera (objeto de clase
          contenida en el modulo DataStructures)
    """

    initial_node = Node(initial_state, None, None)
    expanded = 0
    generated = 0
    nodo_explorado = None
    nuevos_expandidos = None

    explorados = []
    #Cola para los nodos explorados.
    frontier.insert(initial_node)

    while True:


        if frontier.is_empty():
            print("mal")
            return None

        #Se elimina el primer nodo de la frontera y se asigna como nodo actual.
        nodo_explorado = frontier.remove()

        find = find_empty(nodo_explorado.state.bo)
        if not find:

            return (nodo_explorado, expanded, generated)

        #Se añade el nodo explorado a la cola
        explorados.append(nodo_explorado)
        #En la variable nuevos_expandidos  se guarda el nodo expandido.
        nuevos_expandidos = nodo_explorado.expand()

        #Se actualiza expanded.
        expanded = expanded + 1

        for nodo_nuevo in nuevos_expandidos:

            if not frontier.contains(nodo_nuevo) and not explorados.__contains__(nodo_nuevo):
                generated = generated + 1
                nodo_nuevo.g = nodo_nuevo.parentPosition.g +1
                frontier.insert(nodo_nuevo)

    return (None, expanded, generated)
```

Figura 12. Método de búsqueda.

Clase State

Esta clase se utiliza para representar el estado del problema del sudoku. Cada estado contiene una matriz con unos números que representan el tablero del sudoku. En esta clase también sirve para generar los sucesores del estado en cuestión, contando para ello de varias funciones.

La principal sería next_states en la cual primero se busca el primer hueco vacío en el tablero, es decir la matriz del estado actual, mediante el método find_empty que recorre la matriz hasta que encuentra un 0 en una posición.

```
def next_states(self):
    """ Dado un estado de tablero comprueba los posibles siguientes estados
    apartir de este. Los que sean válidos los devuelve en un string junto a
    la acción que ha generado ese nuevo estado """
    new_states = []
    row=0
    col=0
    find = find_empty(self.bo)
    if not find:
        print("error")
    else:
        row, col = find

    action = ['1'+str(row)+str(col),'2'+str(row)+str(col),
              '3'+str(row)+str(col),'4'+str(row)+str(col),
              '5'+str(row)+str(col),'6'+str(row)+str(col),
              '7'+str(row)+str(col),'8'+str(row)+str(col),'9'+str(row)+str(col)]
    for i in range(len(action)):
        estado = self.succ(action[i])
        if estado != None:

            new_states.append([estado, action[i]])

    return new_states
```

Figura 13. Método new_states.

```
def find_empty(bo):
    """ Encuentra el primer hueco vacio en la matriz """
    for i in range(len(bo)):
        for j in range(len(bo[0])):
            if bo[i][j] == 0:
                return (i, j)  # row, col
```

Figura 14. Método find_empty.

Después se tienen en cuenta los 9 números posibles en la posición vacía utilizando el método succ donde se crea una copia de la matriz actual para ir comprobando con el método valid, que comprueba según las normas del sudoku si una posición es válida o no, las diferentes opciones y si son correctas las devuelve.

```
def succ(self, action):
    """ Comprueba si un posible sucesor es válido o no """
    num = int (action[0])
    row = int (action[1])
    col = int (action[2])
    bo1 = copy.deepcopy(self.bo)

    if valid(bo1, num, (row, col)):
        bo1[row][col] = num
        estado = SudokuState(bo1)



        return estado
    return None
```

Figura 14. Método succ.

```
def valid(bo, num, pos):
    """ Comprueba si la posicion elegida , en el tablero concreto, cumple las normas del sudoku """
    # Comprueba fila
    for i in range(len(bo[0])):
        if bo[pos[0]][i] == num and pos[1] != i:
            return False

    # Comprueba Columna
    for i in range(len(bo)):
        if bo[i][pos[1]] == num and pos[0] != i:
            return False

    # Comprueba subcuadricula
    box_x = pos[1] // 3
    box_y = pos[0] // 3

    for i in range(box_y*3, box_y*3 + 3):
        for j in range(box_x * 3, box_x*3 + 3):
            if bo[i][j] == num and (i,j) != pos:
                return False

    return True
```

Figura 15. Método valid.

Finalmente, en next_states se devuelven las opciones de estado válidas en una tupla junto a la acción que ha provocado ese estado.

Interfaz

La interfaz de este TFG se ha decidido que sea muy simple dado sus necesidades académicas de ser de fácil uso y claridad máxima. Para ello se ha utilizado el módulo de Pygame.[13]

Pygame es un conjunto de módulos del lenguaje Python que permiten la creación de videojuegos en dos dimensiones de una manera sencilla. Está orientado al manejo de sprites.

Para realizar la interfaz con este módulo, se han utilizado 2 clases simples: grid y cube. Grid es para representar las líneas del tablero y cube cada uno de los cuadrados que conforman las intersecciones de las líneas creadas y su contenido.

En un bucle de ejecución constante del juego se está refrescando constantemente el dibujo de las instancias de estas clases para reflejar cualquier cambio durante la ejecución.

Para controlar fácilmente las acciones de las clases anteriormente explicadas se han implementado 2 botones sencillos.

```
    """ Dibuja toda la interfaz: lineas, subcuadriculas, botones y textos de solucion si se da las condiciones"""
    # Dibuja las lineas
    gap = self.width / 9
    for i in range(self.rows+1):
        if i % 3 == 0 and i != 0:
            thick = 4
        else:
            thick = 1
        pygame.draw.line(win, (0,0,0), (0, i*gap), (self.width, i*gap), thick)
        pygame.draw.line(win, (0, 0, 0), (i * gap, 0), (i * gap, self.height), thick)

    # Dibuja las subcuadriculas
    for i in range(self.rows):
        for j in range(self.cols):
            self.cubes[i][j].draw(win)
    # Color de los botones segun si esta el raton encima o no
    if generar.collidepoint(mouse.get_pos()):
        pygame.draw.rect(win, (155,155,155), generar,5)
    else:
        pygame.draw.rect(win, (0,0,0), generar,5)

    if resolver.collidepoint(mouse.get_pos()):
        pygame.draw.rect(win, (155,155,155), resolver,5)
    else:
        pygame.draw.rect(win, (0,0,0), resolver,5)
    # Texto dentro de los botones
    texto_generar = myfont.render("Generar", True,(0,0,0))
    texto_resolver = myfont.render("Resolver", True,(0,0,0))
```

Figura 16. Método draw.

3.4. Pruebas y resultados

En este apartado se van a comentar los resultados de ejecución de la aplicación del TFG. Se van a analizar la eficacia de los métodos de forma general ya que puede haber variaciones según las características del sudoku.

Por lo general, las búsquedas informadas solucionan el problema con menos nodos generados y expandidos respecto a las búsquedas no informadas.

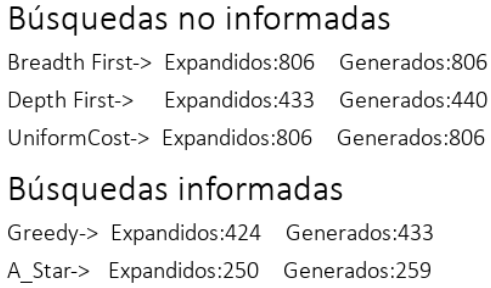


Figura 17. Resultados de ejecución favorables a Informada.

Aunque hay que comentar que dentro de las no informadas la búsqueda Depth First tiene muy buenos resultados en el problema del sudoku, incluso en ocasiones, si el sudoku tiene muchos números iniciales, superando a las búsquedas informadas.

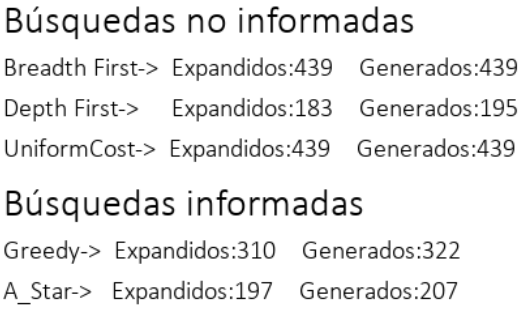


Figura 18. Resultados de ejecución favorables a Depth First.

Respecto a la ejecución, en general cuantos menos números iniciales tiene un sudoku más difícil es por lo que tarda más el programa en ofrecer los resultados al usuario. Lo mismo respecto a la comprobación de número de soluciones al generar el sudoku, cuanto más difícil más tiempo tarda en generarlo.

3.5. Tecnologías utilizadas

Python

Python es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. A diferencia de otros lenguajes como Java o .NET, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en Python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado interpretador, por lo que no es necesario "traducirlo" a lenguaje máquina.

Python es un lenguaje sencillo de leer y escribir debido a su alta similitud con el lenguaje humano. Además, se trata de un lenguaje multiplataforma de código abierto y, por lo tanto, gratuito, lo que permite desarrollar software sin límites. Con el paso del tiempo, Python ha ido ganando adeptos gracias a su sencillez y a sus amplias posibilidades, sobre todo en los últimos años, ya que facilita trabajar con inteligencia artificial, big data, machine learning y data science, entre muchos otros campos en auge.

Figura 19. Logo de Python.

Aptana Studio 3

Aptana Studio es un entorno de desarrollo integrado de software libre basado en eclipse y desarrollado por Aptana, Inc., que puede funcionar bajo Windows, Mac y Linux y provee soporte para lenguajes como: PHP, Python, Ruby, CSS, Ajax, HTML y ActionScript 3.0 (Adobe AIR). Tiene la posibilidad de incluir complementos para nuevos lenguajes y funcionalidades. Los desarrolladores lo definen como el IDE de desarrollo web de código abierto más potente.

Figura 20. Logo de Aptana Studio.

El proceso a llevar a cabo por el alumno con estas tecnologías para realizar el correcto aprendizaje sería el siguiente.

Figura 20. Proceso de aprendizaje.

4. Accesibilidad y Usabilidad

La accesibilidad de una aplicación es su capacidad para que todas las personas puedan utilizar sus funciones y acceder a su contenido independientemente de sus capacidades técnicas, físicas o cognitivas.

Existen diferentes pruebas para analizar la accesibilidad de una aplicación, pero este TFG al ser diferente a una aplicación común no puede realizarlos. Ya que se trata de un proyecto de carácter académico, su accesibilidad se debe probar con alumnos de la titulación del Grado de Ingeniería Informática. Por lo cual queda para realizar en el siguiente curso con los alumnos titulados en la asignatura de IA en el primer cuatrimestre.[6]

Respecto a la usabilidad se va a analizar desde el punto de vista de los 10 principios de Nielsen los cuales se formularon en 1995.

Jakob Nielsen es un gurú de la usabilidad web reconocido en todo el mundo. Tanto es así, que Nielsen ha sido apodado como padre, rey e incluso papa de la usabilidad en internet y creador de los famosos principios de usabilidad de Jakob Nielsen.

Figura 21. Los 10 principios de Nielsen.

1. Visibilidad del estado del sistema

El sistema (web, aplicación o cualquier otro producto digital) debe siempre mantener informado al usuario de lo que está ocurriendo.

Sudoku Game, al contener una interfaz sencilla, muestra en todo momento el estado del problema y las opciones disponibles al usuario. Durante la ejecución, se va informando de todos los caminos y posibilidades tenidos en cuenta por cada método de búsqueda. Esta información, al ser una aplicación con fines académicos, se va mostrando por consola al alumno que está usando el proyecto.

2. Relación entre el sistema y el mundo real

El sitio web o aplicación tiene que utilizar el lenguaje del usuario, con expresiones y palabras que le resulten familiares. Además, la información debe aparecer en un orden lógico y natural.

Toda la información mostrada en la interfaz de Sudoku Game, está en lenguaje común y sencillo para su fácil uso. Además, de cara al estudiante que usará el proyecto, todo lo comentado en el código y en resultados mostrados en consola son términos vistos anteriormente en la propia asignatura de IA.

3. Control y libertad del usuario

En caso de elegir alguna opción del sitio web por error, el usuario agradecerá disponer de una "salida de emergencia" para abandonar el estado no deseado en que se halla. Debe poder deshacer o repetir una acción previamente realizada.

El usuario puede elegir todas las veces que quiera generar un sudoku nuevo antes de resolverlo, además de después de resuelto también. Por otra parte, para evitar problemas, hay un control de errores que evita que se pueda resolver un sudoku si no se ha generado ninguno.

4. Consistencia y estándares

Es importante establecer convenciones lógicas y mantenerlas siempre. El usuario no tiene por qué saber que diferentes palabras, situaciones o acciones significan lo mismo.

Este apartado se aplica más en cuanto al lenguaje usado en los comentarios del código y el estilo de programación seguido.

5. Prevención de errores

Ayuda al usuario a que no caiga en un error.

Como se ha comentado anteriormente, para evitar problemas hay un control de errores que evita que se pueda resolver un sudoku si no se ha generado ninguno.

6. Reconocimiento antes que recuerdo

Debemos hacer visibles acciones y opciones para que el usuario no tenga que recordar información entre distintas secciones o partes del sitio web o aplicación.

En esta aplicación, al contar con solo dos posibilidades, y estando marcadas por dos botones disponibles en todo momento, se hace fácil para el usuario reconocer las opciones que tiene.

7. Flexibilidad y eficiencia de uso

Los aceleradores o atajos de teclado, por ejemplo, pueden hacer más rápida la interacción para usuarios expertos, de tal forma que el sitio web o aplicación sea útil tanto para usuarios básicos como avanzados.

Este apartado en principio no es necesario cumplirlo, ya que el uso de esta aplicación de cara a la interfaz es muy sencillo y no necesita de atajos ni opciones avanzadas.

8. Estética y diseño minimalista

Las páginas no deben contener información innecesaria. Cada información extra compite con la información relevante y disminuye su visibilidad.

La información de resultados en Sudoku Game solo se muestra al resolver un sudoku, al generar uno nuevo se quita los resultados de la operación anterior.

9. Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores

Los mensajes de error se deben entregar en un lenguaje claro y simple, indicando en forma precisa el problema y sugerir una solución constructiva al problema.

El control de errores comentado anteriormente, se muestra al usuario con una ventana emergente que le explica concretamente cómo solucionar el error.

10. Ayuda y documentación

Aunque es mejor que el sitio web o aplicación pueda ser usado sin ayuda, puede ser necesario proveer cierto tipo de ayuda. En este caso, la ayuda debe ser fácil de localizar, especificar los pasos necesarios y no ser muy extensa.

La ayuda asociada con esta aplicación va relacionada con su uso académico. Ya que el usuario objetivo va a ser un alumno que tenga que completar un boletín de prácticas, se le explicara en este mismo los pasos a seguir para poder llegar a una versión similar o parecida a la presentada en este TFG.

El TFG de Sudoku Game cumple la mayoría de los puntos de Usabilidad de Nielsen, exceptuando los que no se aplican a un proyecto de estas características especiales, al ser un TFG educativo para aprender a desarrollar métodos de IA.

5. Licencia software y documental

Copyright © 2022, Ignacio Gutiérrez Villar
Este documento, así como sus anexos son liberados bajo los términos de una licencia GFDL.
Se permite la redistribución, copia y/o modificación de estos documentos bajo los términos de la GNU Free Documentation License, versión 1.3 o cualquier versión posterior publicada por la Free Software Foundation.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation.

Tampoco hay que olvidar que este documento, por defecto, está al amparo de la licencia 7.3, por su inclusión en el Repositorio Institucional de Documentos de la Universidad de Zaragoza: ZAGUAN

El programa liberado es software libre: puedes redistribuirlo o modificarlo bajo los términos de la GNU General Public License publicada por la Free Software Foundation en su versión 3 o cualquier versión posterior.
Este programa es distribuido con el propósito de que sea útil, pero CON NINGUNA GARANTÍA, incluso sin garantía de COMERCIALIZACIÓN o APTITUD PARA CUALQUIER PROPÓSITO CONCRETO.
This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

ESTE SOFTWARE ES SUMINISTRADO POR IGNACIO GUTIÉRREZ VILLAR Y CUALQUIER GARANTÍA EXPRESA O IMPLÍCITA, INCLUYENDO, PERO NO LIMITANDO A: LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN Y APTITUD PARA UN PROPÓSITO PARTICULAR, SON RECHAZADAS. EN NINGÚN CASO IGNACIO GUTIERREZ VILLAR O COLABORADORES SERÁN RESPONSABLES POR NINGÚN DAÑO DIRECTO, INDIRECTO, INCIDENTAL, ESPECIAL, EJEMPLAR O CONSECUENCIAL (INCLUYENDO, PERO NO LIMITADO A: LA ADQUISICIÓN O SUSTITUCIÓN DE BIENES O SERVICIOS, LA PÉRDIDA DE USO, DE DATOS O DE BENEFICIOS; O INTERRUPCIÓN DE LA ACTIVIDAD EMPRESARIAL) O POR CUALQUIER TEORÍA DE RESPONSABILIDAD, YA SEA POR CONTRATO, RESPONSABILIDAD ESTRICTA O AGRAVIO (INCLUYENDO NEGLIGENCIA O CUALQUIER OTRA CAUSA) QUE SURJA DE CUALQUIER MANERA DEL USO DE ESTE SOFTWARE, INCLUSO SI SE HA ADVERTIDO DE LA POSIBILIDAD DE TALES DAÑOS.

6. Conclusiones y trabajo futuro

Respecto al TFG en conjunto, las conclusiones a las que he llegado tras completarlo son positivas. Al principio resultó un reto plantear un TFG que fuese a ser usado en las aulas. Pero es cierto que tras pensarlo y consultar documentación relacionada con la asignatura de IA fue más fácil ir dando forma al TFG. También ayudó la formación durante la titulación en cuanto a la materia de IA, lo cual ha llevado a mi satisfacción con el resultado de la aplicación y su documentación en forma de este TFG.

La parte más pura de programación del TFG me ha ayudado a poner en práctica conocimientos aprendidos durante estos años de universidad, concretamente de las asignaturas de Tecnología de la Programación, Programación I y II, Teoría de la Computación, Estructuras de Datos y Algoritmos y por supuesto Inteligencia Artificial.

También he aprendido la importancia de la organización y estructurar bien un proyecto, visto en la asignatura Ingeniería del software, para cumplir plazos y objetivos. Así como dividir grandes problemas por afrontar en pequeños problemas, y resolverlos con buenas prácticas de diseño. Con todo ello, en definitiva, presentar un TFG del que estar orgulloso.

En cuanto al trabajo futuro a realizar primero estaría realizar las pruebas de accesibilidad con los alumnos de la asignatura de IA, como se comentó anteriormente en el apartado de Accesibilidad y Usabilidad.

Para continuar quedaría por implementar más métodos de búsqueda tanto informados como no informados y analizar sus resultados en el problema del sudoku.

También, para añadir más variedad se podrá implementar en un futuro seleccionar diferentes tipos de sudoku, ahora solo está disponible el clásico 9x9, que tengan normas especiales y diferentes grados de dificultad.

Para terminar, se podrá trabajar en un futuro en ampliar el conocimiento sobre el aprendizaje basado en el pensamiento, realizando un trabajo de investigación académica, respecto a este tipo de enseñanza en el ámbito de la IA, que podría dar lugar a alguna publicación de revista relevante en el ámbito.

7. Referencias Bibliográficas

[1] CAMPOS, JAVIER, 2022, Estructuras de Datos y Algoritmos (EDA). Webdiis.unizar.es [online]. 2022. [Accessed 15 June 2022]. Available from: http://webdiis.unizar.es/asignaturas/EDA/

[2] CHARLYRED70, 2022, ¿Qué es Heurística? PROGRAMACIÓN PARA TODOS [online]. 2022. [Accessed 15 June 2022]. Available from: https://programacionparatodossite.wordpress.com/2016/10/20/que-es-heuristica/

[3] Definición de ingeniería de software, 2022. definicion.de [online]. Available from: https://definicion.de/ingenieria-de-software/

[4] JJPELEATO, 2022, 2.4 Backtracking - Programación, refactoriza tu mente. Docs.jjpeleato.com [online]. 2022. [Accessed 15 June 2022]. Available from: https://docs.jjpeleato.com/algoritmia/backtracking

[5] Lambda Functions, 2022. Sequoiatree.github.io [online]. Available from: https://sequoiatree.github.io/lambda-functions.html

[6] MORA, SERGIO, 2022, Accesibilidad Web: española. Accesibilidadweb.dlsi.ua.es [online]. 2022. [Accessed 15 June 2022]. Available from: http://accesibilidadweb.dlsi.ua.es/?menu=espanola

[7] ONU, 2022, Educación - Desarrollo Sostenible. Desarrollo Sostenible [online]. 2022. [Accessed 15 June 2022]. Available from: https://www.un.org/sustainabledevelopment/es/education/

[8] Problema de satisfacción de restricciones - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online]. Available from: https://es.wikipedia.org/wiki/Problema_de_satisfacci%C3%B3n_de_restricciones

[9] RUMBAUGH, JAMES, JACOBSON, IVAR and BOOCH, GRADY, 2007, El lenguaje unificado de modelado. Madrid: Pearson Educación.

[10] RUSSELL, STUART J and NORVIG, PETER, 2010, Artificial intelligence: a modern approach. 3. Boston: Pearson.

[11] SOMMERVILLE, IAN, 2009, Software engineering. 7. Madrid: Pearson Educación.

[12] Sudoku - Wikipedia, la enciclopedia libre, 2022. Es.wikipedia.org [online]. Available from: https://es.wikipedia.org/wiki/Sudoku

[13] TECH WITH TIM, TIM, 2022, Sudoku solver backtracking. Techwithtim.net [online]. 2022. [Accessed 15 June 2022]. Available from: https://www.techwithtim.net/tutorials/python-programming/sudoku-solver-backtracking/

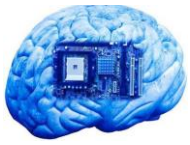
[14] THAILAPPAN, DHANYA, 2022, An Introduction to Problem-Solving using Search Algorithms for Beginners. Analytics Vidhya [online]. 2022. [Accessed 15 June 2022]. Available from: https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-problem-solving-using-search-algorithms-for-beginners/

[15] VASILIU, ADA, 2022, Uninformed search algorithms in Python - Cyluun inside. Cyluun.github.io [online]. 2022. [Accessed 15 June 2022]. Available from: https://cyluun.github.io/blog/uninformed-search-algorithms-in-python

8. Anexos

Enunciado de boletín de prácticas

Práctica 2: Resolución de problemas mediante búsqueda



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Autores: Piedad Garrido e Ignacio Gutiérrez
Grado en Ing. Informática. Inteligencia Artificial 2022/2023

Contenido

Objetivos de la práctica 3
Búsqueda no informada e informada 3
Problema a resolver 5
Ficheros de la práctica 7
Entrega de la práctica 8

Objetivos de la práctica

Esta práctica se centrará en la resolución básica de problemas mediante búsqueda por expansión de estados. Se probarán diferentes estrategias tanto informadas como no informadas y, se estudiará su eficiencia a la hora de resolver problemas en entornos deterministas, completamente observables y estáticos. Gran parte de los juegos para un solo jugador podrían caer en esta categoría, pero también otros sistemas en los que se puedan definir de forma inequívoca los diferentes estados del sistema y las transiciones entre los mismos mediante una serie finita de acciones.

Búsqueda no informada e informada

Los problemas de búsqueda siempre comienzan con un estado inicial del sistema, y se centran en encontrar un estado final que satisface ciertas condiciones. Por tanto, debemos definir para cada problema los siguientes parámetros:

- El estado inicial del sistema

- Las posibles acciones que pueden producirse desde cada uno de los posibles estados del sistema

- El resultado de dichas acciones, es decir, a qué estado nos conduciría una acción dado un estado dado

- Una función de coste, que asigna un coste a cada serie de acciones. Normalmente se representa como g(n)

- Una prueba para determinar si un estado es el final (también llamado meta u objetivo)

El esquema general de un algoritmo de búsqueda que usaremos sera el de GRAPH-SEARCH. La única diferencia entre TREE-SEARCH y GRAPH-SEARCH es que GRAPH-SEARCH incorpora una lista de estados visitados (explorados) para evitar repeticiones, por lo que es más eficiente especialmente en entornos con multitud de estados. Por ello, usaremos este algoritmo como base para la práctica:

```
function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
            only if not in the frontier or explored set
```

Existen diferentes variantes del mismo, todas ellas basadas en diferencias a la hora de elegir un nodo a expandir de la frontera de estados visitados. Podemos hacer una primera división entre algoritmos de búsqueda no informada, o informada:

• **Búsqueda no informada (ciega):** las estrategias no disponen de información adicional sobre estados más allá de la proporcionada en la definición del problema, por lo que sólo pueden expandir estados y distinguir entre estados objetivos y no-objetivos. Dentro de esta categoría se distinguen:

  o **Búsqueda primero en anchura (breadth-first):** se expanden primero los estados de un nivel antes de expandir los del estado siguiente.

  o **Búsqueda primero en profundidad (depth-first):** se expanden primero los estados de mayor nivel (los más profundos).

  o **Búsqueda en coste uniforme (uniform cost):** se expanden primero los estados con menor coste acumulado de acciones para llegar a él (función g).

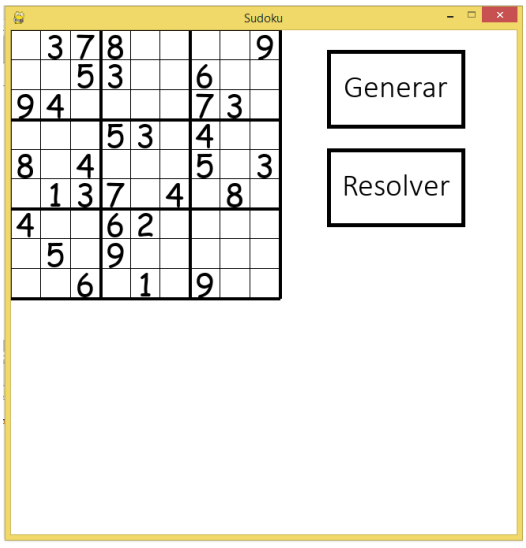
• **Búsqueda informada (heurística):** se utiliza información propia del problema más allá de la proporcionada en la definición. Se utiliza una función heurística, normalmente denominada h(n), que estima el coste del camino más corto desde el estado actual hasta el objetivo. Para que la heurística sea admisible, nunca puede indicar un coste mayor que el real para algún estado. En esta categoría se incluirían:

  o **Búsqueda voraz, avariciosa o primero el mejor (greedy or best-first):** se expanden primero los estados con el menor valor de la función heurística, h.

  o **Búsqueda A*:** se expanden primero los estados con menor función f, obtenida como el valor de la suma del coste para llegar al estado más la estimación de alcanzar el estado objetivo, es decir, f = g + h.

Problema a resolver

Para aplicar las estrategias de búsqueda, resolveremos el juego del Sudoku. El objetivo del sudoku es rellenar una cuadrícula de 9 × 9 celdas dividida en subcuadrículas de 3 × 3 con los números del 1 al 9 partiendo de algunos números ya están puestos en algunas de las celdas. Para completar el sudoku un número no se debe repetir en una misma fila, columna o subcuadrícula.
El estado inicial del problema será cada vez diferente al tener un generador de sudokus ya proporcionado en la práctica.
El estado final será que todas las celdas están rellenadas por un número y cumpliendo las normas del sudoku.



Ficheros de la práctica

La práctica está compuesta por los siguientes ficheros:

• datastructures.py: contiene estructuras de datos útiles en Python para la realización de los algoritmos (Stack, Queue, PriorityQueue).

• search.py: contiene la estructura básica del proceso de búsqueda no informada e informada. Sin embargo, algunas de las funciones no tienen el código necesario para que el algoritmo funcione correctamente. Por tanto, cada alumno debe rellenar el código de las siguientes funciones:

- o uninformed_search
- o breadth_first
- o depth_first
- o uniform_cost
- o informed_search
- o greedy
- o a_star
- o h1 u otras funciones heurísticas.

• state.py: contiene la clase que representa cada estado del problema. Cada alumno debe rellenar el código de las siguientes funciones:

- o succ
- o valid
- o is_empty
- o next_states

• sudoku.py: contiene el generador de sudokus mediante backtracking.

• main.py: proporciona una interfaz gráfica para poder ver la solución encontrada con alguno de los algoritmos de búsqueda, las llamadas a los métodos de búsqueda y generar sudokus.

Entrega de la práctica

Deberán completare las siguientes tareas para considerar la práctica entregada:

1. Completar el código Python necesario para que los algoritmos de búsqueda no informada e informada funcionen correctamente, siguiendo el algoritmo GRAPH-SEARCH. Además, deberán proponerse, al menos 2 funciones heurísticas diferentes y admisibles para ser utilizadas en los algoritmos de búsqueda informada. La eficiencia de las funciones heurísticas formará parte de la evaluación de la práctica.

2. Estudiar el funcionamiento de los algoritmos en función de su:

a. Optimalidad: un algoritmo es óptimo si es capaz de encontrar la solución con un menor coste asociado. En este caso, el coste está relacionado con el número de pasos desde el estado inicial hasta el final.

b. Eficiencia: relacionada con el número de estados que es necesario comprobar para alcanzar la solución. Debe compararse la longitud de las soluciones encontradas y el número de nodos expandidos y generados por cada algoritmo. Discutir los resultados y comparar los resultados obtenidos en la práctica con las características teóricas de cada algoritmo.

3. La entrega se llevará a cabo a través de la tarea habilitada en la plataforma Moodle.