Trabajo TP6-1

Lenguajes de Reglas,
Propagación de restricciones, y juegos

Inteligencia Artificial

3º curso, Grado de Ingeniería en Informática

10 de diciembre de 2021

Leticia Sánchez Romero, 760549

Índice de contenidos

1. CLIPS - Fichas

En esta tarea se pedía implementar un A* y la heurística de cuenta el número de fichas descolocadas. Para realizar la tarea se ha utilizado de modelo el problema proporcionado como material de clase *puzzle-8Astar.clp*.
Se ha modificado la declaración de nodo, y de estado inicial. Además se modifican los operadores para que estos sean los del juego de las fichas. Se tiene el operador izquierda que lo que hace es mover el hueco hacia la izquierda y el operador derecha que mueve el hueco hacia la derecha.
Se modifica también el módulo solución para que muestre los distintos estados que va teniendo el juego hasta que se llega a la solución final.

A continuación se puede ver el resultado de la ejecución del programa:

```
         CLIPS (6.30 3/17/15)
CLIPS> Loading Selection...

[CSTRCPSR1] WARNING: Redefining defmodule: MAIN
Defining deftemplate: nodo
Defining defglobal: estado-inicial
Defining defglobal: estado-final
Defining deffunction: heuristica
Defining deffacts: nodoIncial
Defining defrule: pasa-el-mejor-a-cerrado-A* +j+j+j
Defining defmodule: OPERADORES
Defining defrule: izquierda +j+j
Defining defrule: derecha +j+j
Defining defmodule: RESTRICCIONES
Defining defrule: repeticiones-de-nodo +j+j+j
Defining defmodule: SOLUCION
Defining defrule: reconoce-solucion +j+j
Defining defrule: escribe-solucion +j+j
CLIPS> (reset)
CLIPS> (run)
La solucion tiene:10 pasos
B H B B V V V
B V B B H V V
B V H B B V V
B V V B B H V
B V V B B V H
B V V H B V B
H V V B B V B
V V H B B V B
V V V B B H B
V V V H B B B
CLIPS>
```

2. CLIPS - casillas

En esta tarea se pide completar el código proporcionado.
Se define el nodo en el que se considera el estado = casilla, que es la casilla en la que se encuentra el jugador, y además para controlar las restricciones del juego y de la búsqueda CU se necesita saber cuántas veces se ha andado o saltado.
Se define el estado inicial; la regla para realizar CU con ayuda de los ejemplos proporcionados como material de clase; los operadores saltar y andar siguiendo las reglas del juego; y la restricción con ayuda del material.

A continuación se puede ver el resultado de la ejecución del programa:

```
         CLIPS (6.30 3/17/15)
CLIPS> Loading Selection...

[CSTRCPSR1] WARNING: Redefining defmodule: MAIN
Defining deftemplate: nodo
Defining deffacts: estado-inicial
Defining defrule: pasa-el-mejor-a-cerrado-CU +j+j+j
Defining defmodule: OPERADORES
Defining defrule: Andar +j+j
Defining defrule: Saltar +j+j
Defining defmodule: RESTRICCIONES
Defining defrule: repeticiones-de-nodo +j+j+j

[CSTRCPSR1] Expected the beginning of a construct.
Defining defmodule: SOLUCION
Defining defrule: encuentra-solucion +j+j
Defining defrule: escribe-solucion +j+j
CLIPS> (reset)
CLIPS> (run)
La solucion tiene 4 pasos
(1) (2) (3) (4) (8)
CLIPS>
```

3. Sudoku

Se han creado las clases proporcionadas como material y además se han implementado las clases *SudokuVariable*, *SudokuConstraint* , y *SudokuApp*.
Las dos primeras se han implementado utilizando de referencia el ejemplo proporcionado.
El main de sudoku se ha implementado con la referencia del ejemplo y utilizando métodos de las clases proporcionadas para mostrar por salida estándar lo que se indica en el guión.

A continuación se puede ver el resultado (la parte final) de la ejecución del programa:

```
SOLUCIÓN:
798635421
126974583
453218679
972586314
564123897
381497256
617352948
835749162
249861735
Sudoku solucionado correctamente
---------
....7..2.
8.......6
.1.2.5...
9.54....8
.........
3....85.1
...3.2.8.
4.......9
.7..6....
SUDOKU INCOMPLETO - Resolviendo
{Cell at [0][7]=2, Cell at [8][1]=7, Cell at [0][4]=7, Cell at [2][1]=1, Cell at [6][7]=8, Cell at [8][4]=6, Cell at [1][0]=8, Cell at [1][8]=6
Time to solve = 0.071 segundos
SOLUCIÓN:
594876123
823914756
617235894
965421378
781653942
342798561
159342687
436587219
278169435
Sudoku solucionado correctamente
+++++++++
Se han resuelto 156 sudokus
```

La parte que se corta en la derecha en la imagen se puede ver completa aquí:

```
{Cell at [0][7]=2, Cell at [8][1]=7, Cell at [0][4]=7, Cell at [2][1]=1, Cell at
[6][7]=8, Cell at [8][4]=6, Cell at [1][0]=8, Cell at [1][8]=6, Cell at [3][2]=5, Cell at
[5][6]=5, Cell at [7][0]=4, Cell at [7][8]=9, Cell at [2][3]=2, Cell at [2][5]=5, Cell at
[3][3]=4, Cell at [5][5]=8, Cell at [6][3]=3, Cell at [6][5]=2, Cell at [3][0]=9, Cell at
[3][8]=8, Cell at [5][0]=3, Cell at [5][8]=1, Cell at [0][0]=5, Cell at [1][3]=9, Cell at
[5][4]=9, Cell at [7][5]=7, Cell at [0][8]=3, Cell at [3][1]=6, Cell at [6][1]=5, Cell at
[3][7]=7, Cell at [2][7]=9, Cell at [4][8]=2, Cell at [3][6]=3, Cell at [3][4]=2, Cell at
[3][5]=1, Cell at [4][1]=8, Cell at [8][0]=2, Cell at [7][1]=3, Cell at [8][6]=4, Cell at
[8][5]=9, Cell at [2][8]=4, Cell at [6][8]=7, Cell at [8][8]=5, Cell at [1][6]=7, Cell at
[0][1]=9, Cell at [0][2]=4, Cell at [0][3]=8, Cell at [0][5]=6, Cell at [0][6]=1, Cell at
[1][1]=2, Cell at [1][2]=3, Cell at [1][4]=1, Cell at [1][5]=4, Cell at [1][7]=5, Cell at
[2][0]=6, Cell at [2][2]=7, Cell at [2][4]=3, Cell at [2][6]=8, Cell at [4][0]=7, Cell at
[4][2]=1, Cell at [4][3]=6, Cell at [4][4]=5, Cell at [4][5]=3, Cell at [4][6]=9, Cell at
[4][7]=4, Cell at [5][1]=4, Cell at [5][2]=2, Cell at [5][3]=7, Cell at [5][7]=6, Cell at
[6][0]=1, Cell at [6][2]=9, Cell at [6][4]=4, Cell at [6][6]=6, Cell at [7][2]=6, Cell at
[7][3]=5, Cell at [7][4]=8, Cell at [7][6]=2, Cell at [7][7]=1, Cell at [8][2]=8, Cell at
[8][3]=1, Cell at [8][7]=3}
```

4. Juego NIM

Para que se pueda elegir el número de cerillas inicial en la caja, cuántas cerillas se pueden coger en cada turno y quien empieza se ha modificado las clases *NimJuegoApp* y *NimJuego*.
En *NimJuego* se ha modificado el constructor para que tenga en cuenta quién es MAX en el estado inicial del juego y el número de cerillas modificado también en el método *getActions(state)*.
Además en la clase *NimJuegoApp* se modifica la interacción por salida estándar con el usuario para poder obtener los datos que se quieren utilizar en la clase mencionada anteriormente.

A continuación se puede ver el resultado de la ejecución del programa:

```
¿Total de cerillas en la caja?
10
¿Cuantas cerillas se pueden quitar en cada turno?
4
¿Empieza computador o humano? (0:Humano, 1:Computador)
0
======================
[0, 10]
Jugador Humano: ¿Cuál es tu acción?
4
La acción elegida es 4
======================
[1, 6]
Jugador computador, elijo acción.
Metricas para Minimax : {expandedNodes=59}
Metricas para Alfa-Beta : {expandedNodes=55}
La acción elegida es 1
======================
[0, 5]
Jugador Humano: ¿Cuál es tu acción?
4
La acción elegida es 4
======================
[1, 1]
Jugador computador, elijo acción.
Metricas para Minimax : {expandedNodes=1}
Metricas para Alfa-Beta : {expandedNodes=1}
La acción elegida es 1
GAME OVER: ¡Gana el humano!
```

5. Tic Tac Toe

Se han estudiado las clases *TitacToeState* y *TicTacToeGame* y se ha implementado la clase TicTacToeApp utilizando como referencia la implementación del juego NIM.

En el programa se juegan 4 partidas, a continuación se puede ver el resultado de sus ejecuciones:

1) Solo máquina MIN MAX

```
MINI MAX con TIC TAC TOE Jugando solo maquina
Juega X
Metrics: {expandedNodes=549945}
Juega O
Metrics: {expandedNodes=59704}
Juega X
Metrics: {expandedNodes=7331}
Juega O
Metrics: {expandedNodes=934}
Juega X
Metrics: {expandedNodes=197}
Juega O
Metrics: {expandedNodes=46}
Juega X
Metrics: {expandedNodes=13}
Juega O
Metrics: {expandedNodes=4}
Juega X
Metrics: {expandedNodes=1}
```

2) Solo máquina ALFA-BETA

```
ALFA-BETA con TIC TAC TOE Jugando solo maquina
Juega X
Metrics: {expandedNodes=30709}
Juega O
Metrics: {expandedNodes=4089}
Juega X
Metrics: {expandedNodes=1519}
Juega O
Metrics: {expandedNodes=220}
Juega X
Metrics: {expandedNodes=97}
Juega O
Metrics: {expandedNodes=32}
Juega X
Metrics: {expandedNodes=13}
Juega O
Metrics: {expandedNodes=4}
Juega X
Metrics: {expandedNodes=1}
```

3) Humano contra máquina MIN MAX

```
MINI MAX DEMO con TIC TAC TOE Jugando contra máquina
====================
- - -
- - -
- - -


Jugador Humano: ¿Cuál es tu acción?
Fila(0-2): 0
Columna(0-2): 0
====================
X - -
- - -
- - -


La máquina juega, y elige:
Acción elegida: Coloco O  ( 1 , 1 )
====================
X - -
- O -
- - -


Jugador Humano: ¿Cuál es tu acción?
Fila(0-2): 0
Columna(0-2): 2
====================
X - X
- O -
- - -


La máquina juega, y elige:
Acción elegida: Coloco O  ( 1 , 0 )
```

```
====================
X O X
- O -
- - -


Jugador Humano: ¿Cuál es tu acción?
Fila(0-2): 2
Columna(0-2): 1
====================
X O X
- O -
- X -


La máquina juega, y elige:
Acción elegida: Coloco O  ( 0 , 1 )
====================
X O X
O O -
- X -


Jugador Humano: ¿Cuál es tu acción?
Fila(0-2): 1
Columna(0-2): 2
====================
X O X
O O X
- X -


La máquina juega, y elige:
Acción elegida: Coloco O  ( 2 , 2 )
```

```
====================
X O X
O O X
- X O


Jugador Humano: ¿Cuál es tu acción?
Fila(0-2): 2
Columna(0-2): 0
GAME OVER:
X O X
O O X
X X O


Ha habido un empate
```

4) Humano contra máquina ALFA-BETA

```
ALFA-BETA DEMO con TIC TAC TOE Jugando contra máquina
=====================
- - -
- - -
- - -


Jugador Humano: ¿Cuál es tu acción?
Fila(0-2): 0
Columna(0-2): 0
=====================
X - -
- - -
- - -


La máquina juega, y elige:
Acción elegida: Coloco O  ( 1 , 1 )
=====================
X - -
- O -
- - -


Jugador Humano: ¿Cuál es tu acción?
Fila(0-2): 1
Columna(0-2): 0
=====================
X - -
X O -
- - -
```

```
La máquina juega, y elige:
Acción elegida: Coloco O  ( 0 , 2 )
=====================
X - -
X O -
O - -


Jugador Humano: ¿Cuál es tu acción?
Fila(0-2): 0
Columna(0-2): 1
=====================
X X -
X O -
O - -


La máquina juega, y elige:
Acción elegida: Coloco O  ( 2 , 0 )
GAME OVER:
X X O
X O -
O - -


¡Gana el computador!
```