الجمهوريَّة العربيَّة السوريَّة

جامعة دمشق

كلية الهندسة المعلوماتية

مشروع مبادئ الذكاء

"Tic-Tac logic"

عبد الله القباني

تقى بكر

يوسف داود

وئام الخطيب

تاريخ التسليم: 2023/6/25.

❖المرحلة الأولى:

سنبدأ بداية بشرح الحقائق والإجرائيات، ومن ثمَّ سنشرح طريقة استخدامهم من أجل اختبار صحة الرقعة من عدمه.

بداية نقوم بتعريف حجم الرقعة:

```
Size(6).
```

المحارف المسموح وضعها ضمن الرقعة بتم التعبير عنها بالإجرائيَّة التاليَّة:

```
valid_symbol(S):- S = x; S = o.
```

ثمَّ نقوم بفرض الحقائق التي تعبر عن الخلايا والرموز الثابتة (أي بناء ال puzzle):

```
%fixed cell
fixed_cell(1,3,x).
fixed_cell(2,3,x).
fixed_cell(3,1,x).
fixed_cell(3,6,x).
fixed_cell(4,1,x).
fixed_cell(4,6,x).
fixed_cell(4,3,o).
fixed_cell(5,2,x).
fixed_cell(5,6,o).
fixed_cell(6,1,o).
fixed_cell(6,5,o).
```

إذا تم وضع هذه الحقائق ضمن level1:- فسيعيد رقعة الأحجية:

```
?- level1.
Game building just a secound:

[[*,*,x,*,*,*],[*,*,x,*,*,*],[x,*,*,*,*,*],[*,*,*,*,*,*],[*,*,*,*,*,*],[*,*,*,*,*,*]]

[[*,*,x,*,*,*],[*,*,x,*,*,*],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,*],[*,*,*,*,*,*]]

|*|*|x|*|*|*|
|*|*|x|*|*|*|
|x|*|*|*|*|x|
|*|*|o|*|*|*|
|*|x|*|*|*|x|
|o|*|*|*|o|*|
```

ونقوم بعد ذلك بتعريف الحقائق التي تعطي الحل بشكل ثابت لإختبار التوابع في المرحلة الأولى:

solve_cell(1,1,x).
solve_cell(1,2,o).
solve_cell(1,4,o).
solve_cell(1,5,x).
solve_cell(1,6,o).
solve_cell(2,1,o).
solve_cell(2,2,x).
solve_cell(2,4,o).
solve_cell(2,5,x).
solve_cell(2,6,o).
solve_cell(3,2,o).
solve_cell(3,3,o).
solve_cell(3,4,x).
solve_cell(3,5,o).
solve_cell(4,1,x).
solve_cell(4,2,o).
solve_cell(4,4,x).
solve_cell(4,5,x).
solve_cell(4,6,o).

```
solve_cell(5,1,o).
solve_cell(5,3,x).
solve_cell(5,4,o).
solve_cell(5,5,o).
solve_cell(6,2,x).
solve_cell(6,3,o).
solve_cell(6,4,x).
solve_cell(6,6,x).
```

إذا تمَّ استدعاء هذه الحقائق ضمن ال level1:- سيقوم بطباعة الرقعة مع حل:

```
Now we will solve the puzzle Row by Row
[[*,o,x,o,x,o],[*,*,x,*,*,*],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[o,x,x,o,o,x],[o,*,*,*,o,*]]

|x|o|x|o|x|o|
|o|x|x|o|x|o|
|x|o|o|x|o|x|
|x|o|o|x|x|o|
|o|x|x|o|o|x|
|o|x|o|x|o|x|
```

نعرف الحقيقة التالية المعبرَّة عن الخليَّة وذلك لسهولة الكود والحل:

```
cell(X, Y, S):- fixed_cell(X, Y, S); solve_cell(X, Y, S).
```

نعرَّف الإجرائيَّة التاليَّة التي تقوم بجلب أسطر الرقعة مرتبة:

```
cells_row(X, Y, [H]):-
    size(Columns), Y =:= Columns,
    cell(X, Y, H).

cells_row(X, Y, [H | T]):-
    size(Columns),
```

```
        Columns > Y,
        cell(X, Y, H),
        Y1 is Y + 1,
        cells_row(X, Y1, T), !.

cells_row(X, Z):-  Y is 1, cells_row(X, Y, Z).
```

وبنفس الطريقة نعرَّف الإجرائيَّة التي تقوم بجلب الأعمدة مرتبة:

```
    cells_column(X, Y, [H | T]):-
        size(Rows),
        Rows > X,
        cell(X, Y, H),
        X1 is X + 1,
        cells_column(X1, Y, T), !.

    cells_column(Y, Z):-
        X is 1,
  cells_column(X, Y, Z).
```

نقوم بتعريف إجرائيَّة الطباعة:

```
    print:-
       size(Rows),
forall(
between(1, Rows, Counter), (cells_row(Counter, Row), print_list(Row))).
```

نقوم بتعريف إجرائيَّة تقوم بحساب عدد محارف ال o, x ضمن الرقعة:

```
    element_count([H|T],E,C):-
        H = E,
        element_count(T,E,C1),
        C is C1+1.
```

```
element_count([H|T],E,C):-
    H\=E,
element_count(T,E,C).
```

وكذلك نعرف الإجرائية التي تقوم بمقارنة عدد محارف ال x مع ال o:

```
 x_equal_o_row([H|T]):-
   element_count([H|T],'x',X),
   element_count([H|T],'y',Y),
  X == Y.
```

نقوم بتعريف الإجرائيَّة التي تختبر وجود ثلاث محارف متتالية من نفس النوع:

```
consecutive([_|T]) :-   consecutive(T).
no_triple([]).
no_triple([H|T]) :-   \+ consecutive([H|T]).
```

نقوم بتعريف الإجرائيَّة التي تتحقق من وجود سطرين متكررين ضمن الرقعة:

```
duplicate([], []).
duplicate([H|T1], [H|T2]):-
   duplicate(T1, T2).

no_duplicate([],[]).
no_duplicate(List1,List2) :-
   \+ duplicate(List1, List2).

no_duplicate_all(0,0).

no_duplicate_all(I,0):-
   I1 is I-1,
   I1 >= 1,
   no_duplicate_all(I1,I1);
   no_duplicate_all(0,0).
```

```
no_duplicate_all(I,J):-
   J1 is J-1,
   J1 is 0 ->
   (   no_duplicate_all(I,0));
   J1 >= 1 ->
   (
   cells_row(I, List1),
   cells_row(J1, List2),
   no_duplicate(List1,List2);
  J1 = 0 ->
   (no_duplicate_all(I,J1)),
  !, fail.
).
```

حيث يقوم تابع ال duplicate باختبار ما اذا كانت سلسلتين متطابقتين فيعيد true، بينما نحن بحاجة قيمة false عند وجود التطابق فتمَّ أخذ الإجرائيَّة no_duplicate على أنها نفي لإجرائيَّة duplicate ، بينما تقوم الإجرائيَّة no_duplicate_all بمسح جميع أسطر الرقعو ومقارنة كل سطر مع بقيَة الأسطر الاخرى بالترتيب.

```
?- no_duplicate([x,x,o],[x,x,o]).
false.

?- no_duplicate([o,x,o],[x,x,o]).
true.
```

للحصول على إجرائيَّة عدم وجود أعمدة مكررة نستدعي الإجرائيَّة التالية:

```
duplicate([], []).
duplicate([H|T1], [H|T2]):-
   duplicate(T1, T2).

no_duplicate([],[]).
no_duplicate(List1,List2) :-
   \+ duplicate(List1, List2).
```

```
no_duplicate_all2(0,0).

no_duplicate_all2(I,0):-
   I1 is I-1,
   I1 >= 1,
   no_duplicate_all2(I1,I1);
   no_duplicate_all2(0,0).

no_duplicate_all2(I,J):-
   J1 is J-1,
   J1 is 0 ->
   (   no_duplicate_all2(I,0));
   J1 >= 1 ->
   (
   cells_column(I, List1),
   cells_column(J1, List2),
   no_duplicate(List1,List2);
   J1 = 0 ->
   (no_duplicate_all(I,J1)),
   !, fail.
   ).
```

ويتم جمع الإجرائيتين السابقتين ضمن إجرائية واحدة no duplicate.

أما في الإجرائيَّة التالية فنتحقق من عدم وجود خلايا فارغة ضمن الرقعة وللقيام بذلك يكفي مسح جمع اسطر الرقعة أو اعمدتها وسنقوم بمسح الأسطر:

```
not_empty([]).
not_empty([H|T]):-
    H = *, !, fail;
    not_empty(T).

check_empty(1).
check_empty(I):-
```

```
I1 is I-1,
cells_row(I1, List),
not_empty(List),
check_empty(I1);
!,fail.
```

التابع **not empty** يتحقق من أنَّ سطر ما لا يحوي خلايا فارغة ويتم التعبير عن أنَّ خلية ما فارغة بالرمز *، أما التابع **check_empty** يمسح جميع الأسطر.

```
Now we will solve the puzzle Row by Row
[[*,*,x,o,x,o],[*,*,x,*,*,*],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,*,x,o,x,o],[o,x,x,o,x,o],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,*,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,*,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,*,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[o,x,x,o,o,x],[o,*,*,*,o,*]]
|x|*|x|o|x|o|
|o|x|x|o|x|o|
|x|o|o|x|o|x|
|x|o|o|x|x|o|
|o|x|x|o|o|x|
|o|x|o|x|o|x|

there is cells empty
```

تم مراعاة السهولة للمسخدم فكل إجرائيَّة من الإجرائيات السابقة التي تقوم باختبار صحة الحل هي إجرائيَّة تعيد **True** أو **false** وعند طباعة أي منها بإمكاننا إرسال

```
Now we will solve the puzzle Row by Row
[[*,o,x,o,o,o],[*,*,x,*,*,*],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,o,o],[o,x,x,o,x,o],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,o,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,o,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,o,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[o,x,x,o,o,x],[o,*,*,*,o,*]]
|x|o|x|o|o|o|
|o|x|x|o|x|o|
|x|o|o|x|o|x|
|x|o|o|x|x|o|
|o|x|x|o|o|x|
|o|x|o|x|o|x|

there is tripple sympole
```

رسالة للمستخدم توضح نوع الخطأ الموجود في الحل على سبيل المثال في حال وجود تكرار يقوم بإرسال رسالة أنه يوجد ثلاث محارف متتالية وتعيد false.

أما في حال أنَّ الحل صحيح فيعيد رسالة بأنَّ الحل صحيح تماماً ولا يوجد أخطاء مثل:

```
Now we will solve the puzzle Row by Row
[[*,o,x,o,x,o],[*,*,x,*,*,*],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[*,x,*,*,*,x],[o,*,*,*,o,*]]
[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[o,x,x,o,o,x],[o,*,*,*,o,*]]
|x|o|x|o|x|o|
|o|x|x|o|x|o|
|x|o|o|x|o|x|
|x|o|o|x|x|o|
|o|x|x|o|o|x|
|o|x|o|x|o|x|
puzzle solved correctly
true
```

❖ المرحلة الثانيَّة:

يتم توليد الحلول ديناميكياً وفق الإجرائيات التالية:

```
:-dynamic solve_cell/3.

fill(X, Y, S):- valid_symbol (S), size(Size), Size >= X, Size
>= Y, solve_cell(X, Y, S1), S1 \= S, retract(solve_cell(X, Y,
S1)), fail.
fill(X, Y, S):- solve_cell(X, Y, S), !.
fill(X, Y, S):- assert(solve_cell(X, Y, S)), !.
```

تقوم هذه الإجرائيَّة بطباعة حلول ديناميكيَّة وفق الآليَّة التالية:

1. يأخذ إحدثيات الخليَّة والرمز الذي يريد إضافته.
2. يقوم بإضافتها ديناميكيَّاً على قاعدة المعطيات باستخدام assert.

3. يتحقق أنَّ الرمز المعطى S هو إما x,o وإلا لا يقوم بإضافة الخليَّة، كما أنه يتحقق في السطر الأول أنَّ الإحداثيات ضمن سعة الرقعة وإلا لا يقوم بالإضافة، كما أنه يتحقق في حال تكرر عمليَّة الإضافة على نفس الموقع بنفس الرمز فلا يقوم بالإضافة أيضاً.

التنفيذ الكلي:

```
?- level1.
Game building just a secound:

[[*,*,x,*,*,*],[*,*,x,*,*,*],[x,*,*,*,*,*],[*,*,*,*,*,*],[*,*,*,*,*,*],[*,*,*,*,*,*]]

[[*,*,x,*,*,*],[*,*,x,*,*,*],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,*],[*,*,*,*,*,*]]

|*|*|x|*|*|*|
|*|*|x|*|*|*|
|x|*|*|*|*|x|
|*|*|o|*|*|*|
|*|x|*|*|*|x|
|o|*|*|*|o|*|

Now we will solve the puzzle Row by Row

[[*,o,x,o,x,o],[*,*,x,*,*,*],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]

[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,*,*,*,*,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]

[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[*,*,o,*,*,*],[*,x,*,*,*,x],[o,*,*,*,o,*]]

[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[*,x,*,*,*,x],[o,*,*,*,o,*]]

[[*,o,x,o,x,o],[o,x,x,o,x,o],[x,o,o,x,o,x],[x,o,o,x,x,o],[o,x,x,o,o,x],[o,*,*,*,o,*]]

|x|o|x|o|x|o|
|o|x|x|o|x|o|
|x|o|o|x|o|x|
|x|o|o|x|x|o|
|o|x|x|o|o|x|
|o|x|o|x|o|x|

puzzle solved correctly

true
```

- مع جزيل الشكر -