جامعة دمشق

الهندسة المعلوماتية

مبادئ الذكاء الصنعي

تقرير مشروع العملي

تقدمة الطلاب:

أحمد حسان الزعبي

أنس مهند الشققي

عبادة محمد أمير المالح

محمد نور محمد بسام قصقص

2021-2022

➢ الفهرس

- الفهرس: 2
- تمثيل المسألة: 3
  - الحقائق المستخدمة ضمن الحل: 3
  - طريقة تقسيم ملفات المشروع: 4
- أولاً: التحقق من كون رقعة ما تمثل حلاً صحيحاً لمسألة: 5
  - التهيئة 5
  - الإجرائية الرئيسية في البرنامج: 5
  - الكود كاملاً 8
- ثانياً: إيجاد حل للمسألة غير المحلولة: 10
  - التهيئة 10
  - الإجرائية الرئيسية 10
  - استراتيجيات الحل التكرارية المستخدمة 11
  - الإجرائيات المساعدة المستخدمة ضمن الإجرائيات السابقة 13
  - الكود كاملاً 15
- مثال عملي 18
  - نتيجة checker 19
  - نتيجة solver 19
- تقسيم العمل بين أعضاء الفريق: 21
  - قسم ال checker 21
  - قسم الـ solver 21

➢ تمثيل المسألة:

• الحقائق المستخدمة ضمن الحل:

تمثيل مكونات المسألة يكون باستخدام حقائق ثابتة بالإضافة إلى حقائق ديناميكية لإمكانية تحقيق الحل.

في الجدول التالي مجموعة الحقائق المستخدمة في تمثيل المسألة وتحقيق الحل:

م	اسم مكون المسألة	الحقيقة Fact المعبرة عنها
1	أبعاد الرقعة	`size(Rows, Columns).`
2	السطر	`row(R).`
3	العمود	`column(C).`
4	خلية الحائط المظللة	`wall(Row, Column).`
5	خلية الحائط التي تحتوي على أرقام	`wall_num(Row, Column, Num).`
6	خلية المصباح	`light(Row, Column).`
7	خلية مضاءة بمصباح (لا تحوي مصباح)	`lighted_cell(Row, Column).`
8	خلية لا يمكن أن تحوي مصباح	`x_cell(Row, Column).`

- الحقائق 2 و 3 هي حقائق ديناميكية تتولد عند تشغيل البرنامج، وهي تستخدم لجلب جميع الخلايا ضمن الرقعة، وذلك لإمكانية المرور على الرقعة كاملة (سيمر معنا لاحقاً كيفية استخدامها).
- الحقيقة رقم 6 هي حقيقة ثابتة في الجزء الأول من البرنامج، حيث أن المطلوب هو التحقق من صحة الحل، أما في الجزء الثاني الذي يتضمن إيجاد حل فهي حقيقة ديناميكية تتولد ضمن البرنامج.
- بالنسبة للحقيقتان 7 و 8 فهما حقائق ديناميكية تستخدم في الجزء الثاني من البرنامج (إيجاد حل للعبة)، بحيث:
  - خلية مضاءة بمصباح: تتولد عند كل خلية تحوي في سطرها أو عمودها على مصباح، ولا تتولد عند خلية المصباح بحد ذاتها.
  - خلية لا يمكن أن تحوي مصباح: تتولد ضمن الخلايا التي وجدنا أثناء الحل أنها من المستحيل أن تحوي مصباحاً، ويكون ذلك بسبب وجودها أمام خلية ذات رقم قد وصل إلى حده.

- **طريقة تقسيم ملفات المشروع:**

المشروع هو عبارة عن ملفين رئيسين هما:

1. **Checker.pl**

يحوي الكود الذي يقوم بالتحقق من مسألة محلولة، فيما إذا كان الحل صحيحاً أم لا، وهو يقابل الجزء الأول من المشروع.

يقوم في بداية الملف بعمل **include** لملف من ملفات الأمثلة ليقوم بجلب الحقائق الثابتة.

2. **Solver.pl**

يحوي الكود الذي يقوم بحل المسألة وتوليد الحقائق الديناميكية المطلوبة.

يقوم في البداية بعمل **include** لملف الـ checker ، وذلك لاستخدامه ضمن الحل.

يحوي المشروع ملفات أمثلة جاهزة للاستخدام، تكون موجودة ضمن المجلد **examples**، كل ملف يحوي الحقائق المعبرة عن مسألة مختلفة، تسهل هذه الملفات تجربة البرنامج على مختلف المسائل بسهولة وسرعة.

➢ أولاً: التحقق من كون رقعة ما تمثل حلاً صحيحاً لمسألة:

يعبر عن هذا الجزء من الحل الملف Checker – كما ذكرنا سابقاً – وسنقوم بعرض جزئياته فيما يلي (للإطلاع على الكود كاملاً انتقل إلى نهاية الفقرة).

- **التهيئة**

الكود التالي يقوم بتهيئة المسألة عن طريق الإجرائية init_checker :

```
:- initialization(init_checker).
init_checker:- retractall(row(_)), retractall(column(_)), size(X, Y),
               set_row(X), set_column(Y).

:- dynamic row/1, column/1.
set_row(0) :- !.
set_row(R):- R > 0, asserta(row(R)), R1 is R - 1, set_row(R1).
set_column(0) :- !.
set_column(C):- C > 0, asserta(column(C)), C1 is C - 1, set_column(C1).
```

- الإجرائية **init_checker** تبدأ بالعمل فور تشغيل البرنامج، وذلك لتعيينها كإجرائية مهيئة (initialization)
- تقوم أولاً بحذف كل الحقائق الديناميكية المخزنة بالذاكرة سابقاً، ثم تقوم بتوليدهم من جديد.
- الحقائق الديناميكية هي row/1, column/1 ، بحيث تعبران عن سطر وعمود، يتم توليد عدد موافق لعدد الأسطر والأعمدة منهما (باستخدام الحقيقة size)، فيتولد من الرقم 1 إلى X و Y.

- **الإجرائية الرئيسية في البرنامج:**

وهي التي تقوم بالتأكد من صحة الحل، بحيث تعيد true في حال كان الحل صحيحاً، و false فيما عدا ذلك:

```
/*
 * Main Rule
 */
solved:- \+any_dimmed_cell, \+any_double_light, \+any_incorrect_count.
```

تتضمن هذه الإجرائية **ثلاث** إجرائيات فرعية:

- `any_dimmed_cell`

تتأكد من كون الرقعة كاملة مضاءة بمصابيح، لا يهم هنا إن كان هناك تكرار في المصابيح أم لا.

```
any_dimmed_cell:- row(R), column(C), \+is_cell_lighted(R, C), !.
is_cell_lighted(R, C):- wall(R, C), !;
                        light(R, C), !;
                        row_items(R, C, RI), \+no_light_in(RI), !;
                        column_items(R, C, CI), \+no_light_in(CI).
```

تقوم هذه الإجرائية بالمرور على كافة خلايا الرقعة، وذلك باستخدام الحقائق الديناميكية (السطر والعمود)، وتعيد true في حال كانت هناك أي خلية **غير مضاءة**، بحيث تتوقف عند أول false قادمة من الإجرائية الجزئية `is_cell_lighted` (وذلك لأننا نقوم بعكسها باستخدام العملية \+(not).

وتقوم الإجرائية `is_cell_lighted` بإرجاع true في إحدى الحالات التالية:

- الخلية الحالية هي خلية جدار (وذلك لأنها لا يمكن أن تضاء)
- الخلية الحالية خلية مصباح
- يوجد في السطر أو العمود (المحدد بحائط أو بجدار الرقعة) مصباح واحد على الأقل.

أما بالنسبة للإجرائية المستخدمة هنا `no_light_in` فهي بالشكل التالي:

```
no_light_in([]) :- !.
no_light_in([[R, C] | T]):- not(light(R, C)), no_light_in(T).
```

فهي تعيد true في حال لم يوجد أي مصباح ضمن القائمة الممررة لها، ثم نقوم بعكسها (not) في الإجرائية الرئيسية للتأكد من وجود مصباح على الأقل.

تبقى الإجرائيتان `row_items` و `column_items` ، فهما من يقومان بتوليد القائمة التي تحوي على إحداثيات الخلايا في السطر أو العمود (المحددة بجدار أو بحدود الرقعة).

```
row_items(R, C, L) :- C1 is C - 1, C2 is C + 1,
            go_left_in_row(R, C1, L1), go_right_in_row(R, C2, L2),
            append(L1, L2, L).
column_items(R, C, L) :- R1 is R - 1, R2 is R + 1,
            go_top_in_column(R1, C, L1), go_bottom_in_column(R2, C, L2),
            append(L1, L2, L).
```

```
go_left_in_row(R, C, []):- wall(R, C), !.
go_left_in_row(_, C, []):- C = 0, !.
go_left_in_row(R, C, [[R, C] | T]):- C1 is C - 1, go_left_in_row(R, C1, T).

go_right_in_row(R, C, []):- wall(R, C), !.
go_right_in_row(_, C, []):- size(_, Y), C > Y, !.
go_right_in_row(R, C, [[R, C] | T]):- C1 is C + 1, go_right_in_row(R, C1, T).

go_top_in_column(R, C, []):- wall(R, C), !.
go_top_in_column(R, _, []):- R = 0, !.
go_top_in_column(R,C,[[R, C]|T]):- R>0, R1 is R-1, go_top_in_column(R1, C, T).

go_bottom_in_column(R, C, []):- wall(R, C), !.
go_bottom_in_column(R, _, []):- size(X, _), R > X, !.
go_bottom_in_column(R,C,[[R, C]|T]):- R1 is R+1, go_bottom_in_column(R1,C, T).
```

تبدأ من الخلية الحالية وتقوم بالمشي يميناً ويساراً في السطر، وفوقاً وتحتاً في العمود، حتى الوصول إلى خلية حائط أو حدود الرقعة.

- any_double_light

تقوم بالتأكد من عدم وجود أي مصباحين في نفس السطر أو العمود المحدد بجدار أو بحدود الرقعة

```
any_double_light:- light(X, Y), row_items(X, Y, L1), \+no_light_in(L1).
any_double_light:- light(X, Y), column_items(X, Y, L2), \+no_light_in(L2).
```

تقوم بالمرور على كافة المصابيح ضمن الرقعة، وتستخدم نفس الإجرائية السابقة no_light_in للتأكد من عدم وجود مصباح آخر في نفس المجال، بالإضافة إلى إجرائيات الحصول على عناصر السطر والعمود.

- any_incorrect_count

تقوم بالتأكد من عدد المصابيح حول الخلايا ذات الأرقام، وذلك بنفس طريقة الإجرائية الأولى، بحيث نبحث عن أي حالة خاطئة

```
any_incorrect_count:- wall_num(X, Y, N), neighbour(X, Y, L),
                      lights_count_in(L, C), C \= N.

neighbour(X, Y, L):- X1 is X - 1, X2 is X + 1,
                     Y1 is Y - 1, Y2 is Y + 1,
                     L = [[X1, Y], [X, Y1], [X2, Y], [X, Y2]].

lights_count_in([], 0) :- !.
```

```
lights_count_in([[X, Y] | T], C):- light(X, Y), !, lights_count_in(T, C1),
                                   C is C1 + 1.
lights_count_in([_ | T], C):- lights_count_in(T, C).
```

الإجرائية **neighbour** تعيد قائمة بجوارات الخلية ذات الرقم.

الإجرائية **lights_count_in** تعيد عدد المصابيح ضمن الخلايا في القائمة.

- الكود كاملاً

```
:- include("examples/ex1.pl").

:- initialization(init_checker).
init_checker:- retractall(row(_)), retractall(column(_)), size(X, Y),
               set_row(X), set_column(Y).

:- dynamic row/1, column/1.
set_row(0) :- !.
set_row(R):- R > 0, asserta(row(R)), R1 is R - 1, set_row(R1).
set_column(0) :- !.
set_column(C):- C > 0, asserta(column(C)), C1 is C - 1, set_column(C1).

/*
 * Main Rule
 */
solved:- \+any_dimmed_cell, \+any_double_light, \+any_incorrect_count.

any_dimmed_cell:- row(R), column(C), \+is_cell_lighted(R, C), !.
is_cell_lighted(R, C):- wall(R, C), !;
                        light(R, C), !;
                        row_items(R, C, RI), \+no_light_in(RI), !;
                        column_items(R, C, CI), \+no_light_in(CI).

any_double_light:- light(X, Y), row_items(X, Y, L1), \+no_light_in(L1).
any_double_light:- light(X, Y), column_items(X, Y, L2), \+no_light_in(L2).

any_incorrect_count:- wall_num(X, Y, N), neighbour(X, Y, L),
                      lights_count_in(L, C), C \= N.
neighbour(X, Y, L):- X1 is X - 1, X2 is X + 1,
                     Y1 is Y - 1, Y2 is Y + 1,
                     L = [[X1, Y], [X, Y1], [X2, Y], [X, Y2]].

no_light_in([]) :- !.
```

```
no_light_in([[R, C] | T]):- not(light(R, C)), no_light_in(T).

lights_count_in([], 0) :- !.
lights_count_in([[X, Y] | T], C):- light(X, Y), !,
                                   lights_count_in(T, C1), C is C1 + 1.
lights_count_in([_ | T], C):- lights_count_in(T, C).

row_items(R, C, L) :- C1 is C - 1, C2 is C + 1,
            go_left_in_row(R, C1, L1), go_right_in_row(R, C2, L2),
            append(L1, L2, L).
column_items(R, C, L) :- R1 is R - 1, R2 is R + 1,
            go_top_in_column(R1, C, L1), go_bottom_in_column(R2, C, L2),
            append(L1, L2, L).

go_left_in_row(R, C, []):- wall(R, C), !.
go_left_in_row(_, C, []):- C = 0, !.
go_left_in_row(R, C, [[R, C] | T]):- C1 is C - 1, go_left_in_row(R, C1, T).
go_right_in_row(R, C, []):- wall(R, C), !.
go_right_in_row(_, C, []):- size(_, Y), C > Y, !.
go_right_in_row(R, C, [[R, C] | T]):- C1 is C + 1, go_right_in_row(R, C1, T).

go_top_in_column(R, C, []):- wall(R, C), !.
go_top_in_column(R, _, []):- R = 0, !.
go_top_in_column(R,C,[[R, C]|T]):- R>0, R1 is R-1, go_top_in_column(R1, C, T).
go_bottom_in_column(R, C, []):- wall(R, C), !.
go_bottom_in_column(R, _, []):- size(X, _), R > X, !.
go_bottom_in_column(R,C,[[R, C]|T]):- R1 is R+1, go_bottom_in_column(R1,C, T).
```

➢ ثانياً: إيجاد حل للمسألة غير المحلولة:

يعبر عن هذا الجزء الملف Solver – كما ذكرنا سابقاً – ، في هذا الجزء لا يوجد لدينا الحقائق light بشكل ثابت، وإنما مهمتنا إيجادها وإضافتها ديناميكياً.

• التهيئة

كما في الجزء الأول، تقوم الإجرائية init_solver بتهيئة البرنامج، تضمين ملف checker، حذف الإجرائيات الديناميكية الموجود مسبقاً في الذاكرة وتهيئة عدد الأسطر والأعمدة، ونقوم بالإضافة إلى ذلك بتحديد علامة x على الخلاية حول الرقم 0، لأن ذلك يكون لمرة واحدة فقط فهذه هي الخطوات الثابتة (غير المكررة) ضمن خوارزمية الحل، أما باقي الإجرائيات فهي تكرارية نقوم بها في كل خطوة.

```
:- include("checker.pl").

:- initialization(init_solver).
init_solver:- retractall(light(_, _)), retractall(lighted_cell(_, _)),
              retractall(x_cell(_, _)), retractall(row(_)),
              retractall(column(_)),
              size(X, Y), set_row(X), set_column(Y),
              \+mark_0s_neighbours_with_x,
              nb_setval(current_solution, ""), nb_setval(temp, "").

:- dynamic light/2, lighted_cell/2, x_cell/2.
```

نقوم أيضاً بتهيئة متحولين سنقوم باستخدامهم لاحقاً هما: current_solution و temp.

```
mark_0s_neighbours_with_x:- wall_num(X, Y, 0), neighbour(X, Y, L),
                            mark_x(L), fail.
mark_x([]):- !.
mark_x([[R, C] | T]):- assert(x_cell(R, C)), mark_x(T).
```

• الإجرائية الرئيسية

هي إجرائية تكرارية، تتكرر إلى أن يختل أحد الشروط.

```
/*
 * Main rule
 */
```

```
solve:- \+invert_solve.
invert_solve:- \+solved, check_solution_duplication, num_neigbour, \+singles,
    \+singles_in_row, \+singles_in_column, mark_num_neighbours_with_x, solve.
```

الإجرائية في السطر الأول تقوم فقط بعكس الإجرائية التكرارية، وذلك ليكون الناتج true عند إمكانية الحل.

استدعاء solve في نهاية الإجرائية يضمن التكرار وعدم التوقف إلا عند شروط التوقف (حيث تعيد false عند حدوثها).

شروط التوقف اثنان:

- أن تكون الرقعة قد أضيئت بالكامل، نعرف ذلك باستخدام الإجرائية solved القادمة من الملف checker.
- أن نصل إلى حالة غير قابلة للحل، نعرف ذلك عن طريق الإجرائية check_solution_duplication، والتي تقوم بمقارنة الحل السابق مع الحل الحالي، إذا تطابق الحلان فهذا يعني أن العملية التكرارية لم تغير شيئاً على الحل الحالي، فهذا يعني أننا قد وصلنا إلى حالة غير ممكنة الحل.

تستخدم هذه الإجرائية الحل المخزن سابقاً لمقارنته مع الحل الحالي، وتولد الحل عن طريق جمع أنواع الخلايا الموجودة حالياً في سلسلة نصية ضمن المتغيرات التي قمنا بتهيئتها أول الملف (في إجرائية التهيئة).

```
check_solution_duplication:-
    nb_getval(current_solution, S), \+get_current_solution,
    nb_getval(temp, C), nb_setval(temp, ""),
    nb_setval(current_solution, C), S \= C.

get_current_solution:- row(R), column(C),
    nb_getval(temp, T), cell_type(R, C, X),
    string_concat(T, X, B), nb_setval(temp, B), fail.

cell_type(R, C, "W"):- wall(R, C), !.
cell_type(R, C, "L"):- light(R, C), !.
cell_type(R, C, "C"):- lighted_cell(R, C), !.
cell_type(R, C, "X"):- x_cell(R, C), !.
cell_type(_, _, "D").
```

• استراتيجيات الحل التكرارية المستخدمة

- num_neigbour

تقوم بوضع مصابيح عند جوارات الخلية ذات العدد، فقط في حال كان عدد الأضواء التي يمكن وضعها يساوي عدد الأماكن الشاغرة.

```
num_neigbour:- \+light_neighbours(4), \+light_neighbours(3),
               \+light_neighbours(2), \+light_neighbours(1).
light_neighbours(N):- wall_num(X, Y, N), neighbour(X, Y, L),
    lights_count_in(L, C), available_cells(L, L1), length(L1, Len),
    N is C + Len, set_lights_in(L1), fail.
```

الإجرائية **light_neighbours** تقوم بالبحث في الخلايا ذات العدد ومقارنة الأماكن الشاغرة مع عدد المصابيح.

الإجرائية **set_lights_in** تقوم بوضع المصباح في الخلايا داخل القائمة بالإضافة إلى تحديد سطر وعمود المصباح بأنها مضاءة (lighted_cell)

- Singles

تقوم بوضع مصابيح في الخلايا الشاغرة المفردة، أي الخلايا التي لا تحوي في سطرها وعمودها على أماكن لوضع مصباح، وبالتالي لا يمكن إضاءتها إلا بوضع مصباح فيها.

```
singles:- empty_cell(R, C),
          row_items(R, C, L1), available_cells(L1, L11), length(L11, 0),
          column_items(R, C, L2), available_cells(L2, L22), length(L22, 0),
          set_lights_in([[R, C]]), fail.
empty_cell(R, C):- row(R), column(C),
          \+wall(R, C), \+lighted_cell(R, C), \+light(R, C), \+x_cell(R, C).
```

الإجرائية **empty_cell** تجلب الخيارات الممكنة لخلية فارغة، ثم نبحث في سطرها وعمودها عن خلايا فارغة أخرى، فإن لم نجد أي خلية (طول القائمة الناتجة 0) فإننا نضع المسباح فيها.

نقوم في هذه الإجرائية بوضع تعليمة fail لكي نبحث في كافة الخيارات الممكنة، ونتيجة لذلك نستخدم تعليمة العكس (not) في الإجرائية الأساسية لكي لا تؤثر على سير التعليمات.

- singles_in_row و singles_in_column

عملها مشابه لسابقتها، إلا أنها تبحث عن الخلايا المحددة بعلامة x حصراً (أي الحقائق x_cell)، ثم تقوم بالبحث في سطرها وعمودها عن خلايا فارغة (يمكن وضع مصباح فيها)، فإن كانت هناك خلية واحدة فقط في السطر أو في العمود (وليس كلاهما) نضع مصباح في تلك الخلية.

```
singles_in_row:- dimmed_x_cell(R, C),
```

```
        row_items(R, C, L1), available_cells(L1, L11), length(L11, 1),
        column_items(R, C, L2), available_cells(L2, L22), length(L22, 0),
        set_lights_in(L11), fail.
singles_in_column:- dimmed_x_cell(R, C),
        row_items(R, C, L1), available_cells(L1, L11), length(L11, 0),
        column_items(R, C, L2), available_cells(L2, L22), length(L22, 1),
        set_lights_in(L22), fail.
dimmed_x_cell(R, C):- row(R), column(C),
        \+wall(R, C), \+lighted_cell(R, C), \+light(R, C), x_cell(R, C).
```

الإجرائية **dimmed_x_cell** تجلب الخيارات الممكنة للخلايا المحددة بإشارة x.

نقوم في هاتين الإجرائيتين بوضع تعليمة fail لكي نبحث في كافة الخيارات الممكنة، ونتيجة لذلك نستخدم تعليمة العكس (not) في الإجرائية الأساسية لكي لا تؤثر على سير التعليمات.

- mark_num_neighbours_with_x

الاستراتيجية الأخيرة، تقوم في نهاية العملية التكرارية بوضع علامات x حول الخلايا ذات العدد، التي قد اكتمل عدد المصابيح حولها.

```
mark_num_neighbours_with_x:- \+x_neighbours(4), \+x_neighbours(3),
                             \+x_neighbours(2), \+x_neighbours(1).
x_neighbours(N):- wall_num(X, Y, N), neighbour(X, Y, L),
                  lights_count_in(L, C), N is C, available_cells(L, L1),
                  mark_x(L1), fail.
```

كما الإجرائيات السابقة، نستخدم fail هنا أيضاً.

- **الإجرائيات المساعدة المستخدمة ضمن الإجرائيات السابقة**

الإجرائية **set_lights_in** تقوم بوضع المصباح في الخلايا داخل القائمة بالإضافة إلى تحديد سطر وعمود المصباح بأنها مضاءة (lighted_cell)

```
set_lights_in([]):- !.
set_lights_in([[R, C] | T]):- light_row_and_column(R, C), set_lights_in(T).

light_row_and_column(R, C):- assert(light(R, C)), row_items(R, C, L1),
             light_cells(L1), column_items(R, C, L2), light_cells(L2).
```

```
light_cells([]):- !.
light_cells([[R, C] | T]):- assert(lighted_cell(R, C)), light_cells(T).
```

الإجرائية **available_cells** إجرائية عودية تأخذ قائمة كوسيط وتعيد قائمة جديدة تحوي فقط الخلايا الفارغة من القائمة المدخلة.

```
available_cells([], []) :- !.
available_cells([[R, C] | T], [[R, C] | T1]) :-
        \+wall(R, C), \+lighted_cell(R, C), \+light(R, C), \+x_cell(R, C),
        size(X, Y), R > 0, R =< X, C > 0, C =< Y, !, available_cells(T, T1).
available_cells([_ | T], L) :- available_cells(T, L).
```

الإجرائية **print** ، تقوم بطباعة القرعة كاملةً، بحيث تأخذ كل نوع من الخلايا محرفاً محدداً يمثلها:

- خلية الحائط ذات الرقم تكون بالشكل NN حيث N تمثل الرقم (مثال 11 للخلية ذات الرقم 1)
- خلية الحائط السوداء تكون بالشكل WW
- خلية المصباح تكون بالشكل LL
- الخلية المضاءة تكون بالشكل –
- الخلية المحددة برمز x تكون بالشكل xx
- الخلية الفارغة عبارة عن فراغين متتابعين

```
print:- row(R), column(C), print_cell(R, C), size(_, Y), C is Y, nl, fail.
print_cell(R, C):- wall_num(R, C, N), !, write(N), write(N), write(' ').
print_cell(R, C):- wall(R, C), !, write('WW ').
print_cell(R, C):- light(R, C), !, write('LL ').
print_cell(R, C):- lighted_cell(R, C), !, write('-- ').
print_cell(R, C):- x_cell(R, C), !, write('xx ').
print_cell(_, _):- write('   ').
```

نقوم بطباعة سطر جديد nl عندما نصل إلى آخر عمود.

مثال على خرج هذه الإجرائية:

```
2 ?- print.
-- LL -- 11 LL -- --
LL 33 LL 11 -- WW LL
-- xx -- xx --    --
WW 00 --    -- WW 00
-- xx --    --    xx
-- 00 -- WW -- 33
LL -- -- WW --
```

- الكود كاملاً

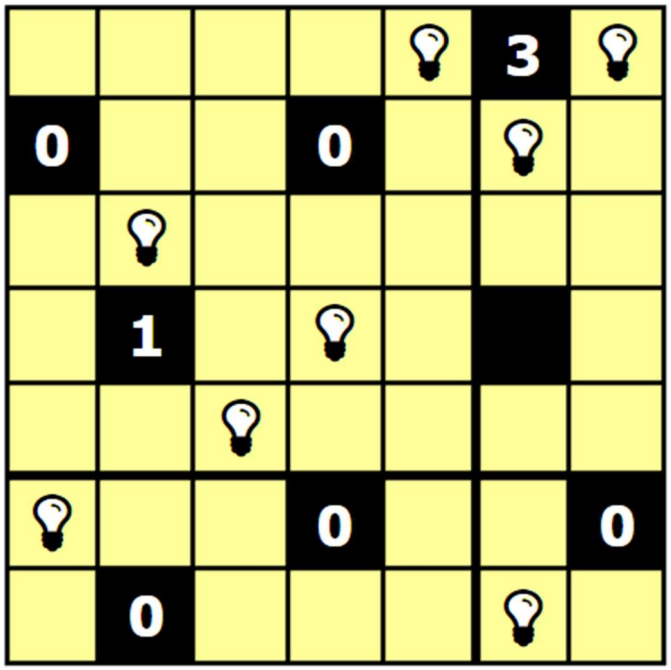
```
:- include("checker.pl").

:- initialization(init_solver).
init_solver:- retractall(light(_, _)), retractall(lighted_cell(_, _)),
              retractall(x_cell(_, _)), retractall(row(_)),
              retractall(column(_)), size(X, Y), set_row(X), set_column(Y),
              \+mark_0s_neighbours_with_x,
              nb_setval(current_solution, ""), nb_setval(temp, "").

:- dynamic light/2, lighted_cell/2, x_cell/2.

/*
 * Main rule
 */
solve:- \+invert_solve.
invert_solve:- \+solved, check_solution_duplication, num_neigbour, \+singles,
    \+singles_in_row, \+singles_in_column, mark_num_neighbours_with_x, solve.

num_neigbour:- \+light_neighbours(4), \+light_neighbours(3),
               \+light_neighbours(2), \+light_neighbours(1).
light_neighbours(N):- wall_num(X, Y, N), neighbour(X, Y, L),
                      lights_count_in(L, C), available_cells(L, L1),
                      length(L1, Len), N is C + Len, set_lights_in(L1), fail.

singles:- empty_cell(R, C),
          row_items(R, C, L1), available_cells(L1, L11), length(L11, 0),
          column_items(R, C, L2), available_cells(L2, L22), length(L22, 0),
          set_lights_in([[R, C]]), fail.
empty_cell(R, C):- row(R), column(C),
          \+wall(R, C), \+lighted_cell(R, C), \+light(R, C), \+x_cell(R, C).

singles_in_row:- dimmed_x_cell(R, C),
          row_items(R, C, L1), available_cells(L1, L11), length(L11, 1),
```

```
          column_items(R, C, L2), available_cells(L2, L22), length(L22, 0),
          set_lights_in(L11), fail.
singles_in_column:- dimmed_x_cell(R, C),
          row_items(R, C, L1), available_cells(L1, L11), length(L11, 0),
          column_items(R, C, L2), available_cells(L2, L22), length(L22, 1),
          set_lights_in(L22), fail.
dimmed_x_cell(R, C):- row(R), column(C),
          \+wall(R, C), \+lighted_cell(R, C), \+light(R, C), x_cell(R, C).

mark_num_neighbours_with_x:- \+x_neighbours(4), \+x_neighbours(3),
                             \+x_neighbours(2), \+x_neighbours(1).
x_neighbours(N):- wall_num(X, Y, N), neighbour(X, Y, L),
          lights_count_in(L, C), N is C, available_cells(L, L1),
          mark_x(L1), fail.

% Helper rules
set_lights_in([]):- !.
set_lights_in([[R, C] | T]):- light_row_and_column(R, C), set_lights_in(T).
light_row_and_column(R, C):- assert(light(R, C)),
          row_items(R, C, L1), light_cells(L1),
          column_items(R, C, L2), light_cells(L2).
light_cells([]):- !.
light_cells([[R, C] | T]):- assert(lighted_cell(R, C)), light_cells(T).

available_cells([], []) :- !.
available_cells([[R, C] | T], [[R, C] | T1]) :-
          \+wall(R, C), \+lighted_cell(R, C), \+light(R, C), \+x_cell(R, C),
          size(X, Y), R > 0, R =< X, C > 0, C =< Y, !, available_cells(T, T1).
available_cells([_ | T], L) :- available_cells(T, L).

mark_0s_neighbours_with_x:- wall_num(X, Y, 0), neighbour(X, Y, L),
          mark_x(L), fail.
mark_x([]):- !.
mark_x([[R, C] | T]):- assert(x_cell(R, C)), mark_x(T).

check_solution_duplication:- nb_getval(current_solution, S),
          \+get_current_solution, nb_getval(temp, C), nb_setval(temp, ""),
          nb_setval(current_solution, C), S \= C.
get_current_solution:- row(R), column(C),
          nb_getval(temp, T), cell_type(R, C, X), string_concat(T, X, B),
          nb_setval(temp, B), fail.
cell_type(R, C, "W"):- wall(R, C), !.
cell_type(R, C, "L"):- light(R, C), !.
cell_type(R, C, "C"):- lighted_cell(R, C), !.
cell_type(R, C, "X"):- x_cell(R, C), !.
cell_type(_, _, "D").
```

```
% Print Rule
print:- row(R), column(C), print_cell(R, C), size(_, Y), C is Y, nl, fail.
print_cell(R, C):- wall_num(R, C, N), !, write(N), write(N), write(' ').
print_cell(R, C):- wall(R, C), !, write('WW ').
print_cell(R, C):- light(R, C), !, write('LL ').
print_cell(R, C):- lighted_cell(R, C), !, write('-- ').
print_cell(R, C):- x_cell(R, C), !, write('xx ').
print_cell(_, _):- write('   ').
```

➢ مثال عملي

ليكن لدينا المثال المحلول التالي:



يمكن التعبير عنه كالتالي:

```
size(7, 7).

wall(1, 6).
wall(2, 1).
wall(2, 4).
wall(4, 2).
wall(4, 6).
wall(6, 4).
wall(6, 7).
wall(7, 2).

wall_num(1, 6, 3).
wall_num(2, 1, 0).
wall_num(2, 4, 0).
wall_num(4, 2, 1).
wall_num(6, 4, 0).
wall_num(6, 7, 0).
wall_num(7, 2, 0).
```

```
light(1, 5).
light(1, 7).
light(2, 6).
light(3, 2).
light(4, 4).
light(5, 3).
light(6, 1).
light(7, 6).
```

- **نتيجة checker**

```
1 ?- solved.
true.
```

فالحل صحيح.

- **نتيجة solver**

نقوم بحذف الحقائق light أولاً ثم نطبق البرنامج:

ملاحظة: سنقوم بتعليق العملية التكرارية والقيام بها يدوياً هنا.

```
1 ?- solve.
-- -- -- -- LL 33 LL
00 -- xx 00 -- LL --
-- LL -- -- -- -- --
xx 11 xx    -- WW --
   xx    xx -- -- --
   xx xx 00 -- -- 00
xx 00 -- -- -- LL --
false.

2 ?- solved.
false.
```

```
3 ?- solve.
-- -- -- -- LL 33 LL
00 -- -- 00 -- LL --
-- LL -- -- -- -- --
-- 11 -- LL -- WW --
-- -- LL -- -- -- --
LL -- -- 00 -- -- 00
-- 00 -- -- -- LL --
false.

4 ?- solved.
true.
```

كما نلاحظ من النتيجة، في المرحلة الأولى لم يكتمل الحل، ثم اكتمل في المرحلة الثانية.

➢ تقسيم العمل بين أعضاء الفريق:

تم تقسيم العمل بشكل متتالي حيث قام اثنان منا باستلام قسم الـ checker والاثنين الآخرين قسم الـ solver.

وبعد انتهاء القسم الأول بادر القسم الثاني بالعمل.

- **قسم الـ checker**

  - عبادة المالح
  - محمد نور قصقص

- **قسم الـ solver**

  - أنس الشققي
  - أحمد الزعبي