الغاية من الجلسة:

توصيف وبرمجة مسألة NQUEEN

شرح المسألة:

مسألة n queens هي مسألة configuration (ليس لدينا حالة هدف وواضحة ولكن نعرف شروط الحالة لتكون هدف) .

في مسألة npuzzel قمنا بالتعبير عن الحالة الابتدائية على شكل مصفوفة ثنائية تحوي أرقام موزعة بطريقة عشوائية من 0 إلى 9 ، باعتبار أن الخانة 0 عي الفراغ الذي يتم تطبيق action عليه مثل (move up , move down ,move left,و move right) .

أما في مسألتنا سيتم التعبير عن الحالة بمصفوفة أحادية من 8 خانات يمثل index الخانة سطر المتواجد فيه الملكة و القيمة الموجودة في الخانة عن العمود المتواجد في الملكة .

الحل هنا هو توزع الملكات على الرقعة بدون وجود أي تضاربات (مائلة ،سطرية أو عمودية).

بحسب الأسلوب الذي سيتم فيه التعبير عن الحالة الرقعة (المصفوفة الأحادية لهذه المسألة) فمن غير الممكن تواجد أي تضارب سطري لأن index في المصفوفة الأحادية هي قيمة unique وبالطبع غير مكررة ونستطيع حساب أو معرفة التضارب العمودي إذا تكررت نفس القسمة في خانتين (وذلك لأن قيمة الخانة تمثل العمود المتواجدة في الملكة كما ذكرنا سابقا).

	0	1	2	3	4	5	6	7
0	Q							
1		Q						
2								Q
3				Q				
4					Q			
5	Q							
6			Q					
7					Q			

0	1	7	3	4	0	2	4

لا بوجد أي تضارب سطري ،تضارب عمودي في العمودين 0 و 4 .

التضارب القطري يكون إذا تواجدت ملكتان على ذات القطر في المصفوفة الثنائية يتم اكتشاف تواجد قيمتين في نفس القطر عن طريق حساب الفرق بين السطرين والعمودين لكل ملكة ونقول أنه تضارب قطري إذا تساو الفرق بينهما (السطرين والعمودين). ولكن في مسألتنا لدينا مصفوفة أحادية) سيتم حساب الفرق بين index (السطرين) وقيم الخانات (الأعمدة).

والان لنبدأ بتوصيف المسالة في البداية سيتم التعريف عن صف الحالة ومن ثم صف action ومن ثم التوجه الى الصف الرئيسي في المسألة .

Class NQueenBoard (state):

```
package examples.NQueens;
import core.State;
public class NQueensBoard implements State {int[] board;
int size;
public NQueensBoard(int[] a) {
this.board = a;

this.size = a.length;}
@Override
public boolean equals(Object s) {
NQueensBoard casted = (NQueensBoard) s;
for (int i = 0; i < size; i++) {
if (this.board[i] != casted.board[i]) {
return false;}}
return true;}
@Override
public boolean isGoal() {
for (int i = 0; i < size; i++) {
if (board[i] == -1) {
return false;}}
return getCoupleAttacks() == 0;}
public String toString() {
String toPrint = "";
for (int j = 0; j < size; j++) {
for (int i = 0; i < size; i++) {
if (this.board[i] == j) {
toPrint += " Q ";}
else{toPrint += " X ";}}
toPrint += "\n";}

return toPrint;}
public int getSize() {
return size;}
public int getCoupleAttacks() {
int numberOfAttacks = 0;
```

```
for (int i = 0; i < size - 1; i++) {
for (int j = i + 1; j < size; j++) {
if (board[i] == board[j]) {
numberOfAttacks++;}
else
if (Math.abs(board[i] - board[j]) == j - i)
{numberOfAttacks++;}}}
return numberOfAttacks;}}
```

Class QueenMovingAction (Action):

```
package examples.NQueens;
import core.IAction;
import core.State;
import java.util.ArrayList;
import java.util.Collection;
public class QueenMovingAction implements IAction {
@Override
public Collection<State> apply(State s) {
NQueensBoard temp = (NQueensBoard) s;
Collection<State> nextStates = new ArrayList<>();
for (int i = 0; i < temp.getSize(); i++) {
int[] t = new int[temp.getSize()];
for (int j = 0; j < temp.getSize(); j++) {
t[j] = temp.board[j];}
t[i] = (temp.board[i] + 1) % temp.getSize();
nextStates.add(new NQueensBoard(t));}

return nextStates;}
@Override
public String getName() {
return "Moving Queen To Next Row";}}
```

يمكن أن نقوم بدلا من تحريك الملكات خطوة في العمود أن نقوم بتحريك الملكات لكل الأسطر الممكنة بالشكل الآتي:

```
for (int i = 0; i < temp.getSize();  i++) {
for (int k = 0; k < temp.getSize();  k++) {
```
نمر هنا على كل أسطر العمود عبر المتحول k
```
int[] t = new int[temp.getSize()];
for (int j = 0; j < temp.getSize();  j++) {
t[j] = temp.board[j];}
t[i] = (temp.board[i]  + k) % temp.getSize();
```
قمنا هنا بإضافة k بدلاً من 1 للمرور على كل أسطر المصفوفة وأخذنا باقي القسمة لكي لا نخرج عن حدود الأسطر
```
nextStates.add(new
 NQueensBoard(t));}}
```

Class GlobalQueensNumberOfNonAttacksHeuristic (Heuristic Function):

```
package examples.NQueens;
import core.HeuristicFunction;
import core.State;
public class GlobalNQueensNumberOfNonAttacksHeuristic
implements HeuristicFunction {
@Override
public double getH(State s) {
NQueensBoard casted = (NQueensBoard)s;
return casted.getCoupleAttacks();}}
```

Class NQueenProblem (Problem):

```
package examples.NQueens;
import core.HeuristicFunction;
import core.IAction;
import core.Problem;
import core.State;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
public class NQueensProblem  extends Problem {private
State initial;
public NQueensProblem(int sz){
initial  = generateBoard(sz);}
public NQueensProblem(State initial){
this.initial = initial;}
@Override
public State getInitialState() {
return initial;}
public void setInitialState(State s){
this.initial =s;}
@Override
public boolean isGoal(State state) {
return state.isGoal();}
@Override
public List<IAction> getActions() {
return new ArrayList(){{add(new QueenMovingAction());}};}
@Override
public double getActionCost(State s, IAction a, State ss) {
return 1;}
@Override
public HeuristicFunction getHeuristicFunction(){
return new GlobalNQueensNumberOfNonAttacksHeuristic();}
public NQueensBoard generateBoard(int size){
int[] start;
start = new int[]{0, 3, 0, 2};
return new NQueensBoard(start);}}
```

يمكن أن نعرف التابع generateBoard بحيث يضع الملكات في مكان عشوائي بالشكل التالي:

```
public NQueensBoard generateBoard(int size){
int[] start = new int[size];
Random gen = new Random(System.nanoTime());
```

نقوم بتعريف مولد عشوائي يولد أرقام عشوائية حسب الوقت الحالي بالواحدة نانوثانية(لكي نحصل على أرقام عشوائية مختلفة عند كل استدعاء)

```
start = new int[size];
for(int i = 0; i < size; i++){
start[i] = -1;}
return new NQueensBoard(start);
```

نختبر الحل في الصف main بالشكل التالي:

```
Problem p = new NQueensProblem(4);
```

قمنا هنا بإنشاء رقعة عدد أسطرها 4 وعدد أعمدتها4

```
Astar astar = new Astar(true);
```

وضعنا True لأن نمط الحل بياني

```
Node astarResult = astar.search(p);
```

قمنا بإرجاع عقدة النهاية التي حصلت عليها خوارزمية*A

```
System.out.println("AStar:\n");
System.out.println(astarResult);
```