CS420 - Wumpus World Project

Group member:
Nguyen Chan Nam - 1751012
Nguyen Nguyen Hoang Thy - 1751040
Dinh Lam Kieu Phuong -1751046

LOGICAL SEARCH ALGORITHM - LEVEL 1

Since the map has a considerable amount of block; in this level, we implement our code mainly by First-order logic algorithm. Some of the simple cases, for example, when the agent enters a safe block, we use Propositional logic algorithm to eliminate possible scenarios for surrounding blocks.

Propositional logic Algorithm

Propositional logic Algorithm is a simple algorithm which combines premises together to create a firm conclusion. For example [1]:
Premise 1: if the left block is safe, the current block is safe
Premise 2: (1,1) is a safe block
Conclusion: (2,1) is a safe block
In our implementation, we only take premise as an absolute truth so as to guarantee the conclusion is usable for further predict

First-order logic Algorithm

First-order logic Algorithm is based on Propositional logic Algorithm. However, if Propositional logic Algorithm only deals with predicate, First-order logic Algorithm denotes atomic sentence to store predicate and terms referring to objects together. Moreover, First-order logic Algorithm deals with universal quantifiers such as exist or for all. In short, each First-order logic Algorithm we use can always be decomposed into Propositional logic Algorithm.
To illustrate the convenience of this algorithm, let's get back to example [1]; From this example, we can obtain similar premise with right, upper, below block into premise 1. To make it shorter, *we define*: $neighborhood - block\ (x) \in$
$\{left\ block\ of\ x, right\ block\ of\ x, upper\ block\ of\ x, below\ block\ of\ x\}$and
$Safe\ (x)$: $block\ x\ is\ safe$. Then,

$$\exists y: y = neighborhood - block\ (x) \wedge Safe\ (y) \rightarrow Safe\ (x)$$

Which is a lot shorter than write down 4 premises to convey the same meaning

Logical background in the project's agent source code

Denote x is the current block, $Neighbor\ (x, y)$means y the left, right, upper or below (if exists) of x, $Safe\ (x)$means x block is safe, $Breeze\ (x)$ means x block is nearby a pit, $Stench\ (x)$means x

block is nearby a Wumpus, $Wump\ (x)$ means x block is a Wumpus, $Pit\ (x)$ means x block is a Pit, $DeadWump\ (x)$ means x block's Wumpus is died

We have some intrinsic prerequisites:

$$\exists y: Neighbor\ (x.y) \wedge Safe\ (y) \rightarrow Safe\ (x)$$
$$\forall y: Neighbor\ (x.y) \wedge Breeze\ (x) \wedge \neg Safe\ (y) \rightarrow Pit\ (y)$$
$$\forall y: Neighbor\ (x.y) \wedge Stench\ (x) \wedge \neg Safe\ (y) \rightarrow Wump\ (y)$$
$$\forall x: DeadWump\ (x) \rightarrow Safe\ (x)$$
$$\forall x: \neg DeadWump\ (x) \wedge Wump\ (x) \rightarrow \neg Safe\ (x)$$
$$\forall x: Pit\ (x) \rightarrow \neg Safe(x)$$

Each plain block, gold block and start block is automatically safe by definition. Other unvisited blocks are automatically unsafe until they are predicted safe or visited.

Advantages and drawbacks

These algorithms brings abundant advantages to the project. Firstly, the algorithm is explainable so that its implemented code is straight-forward. Because the prerequisites for setting next step are all confirmed, the agent is definitely safe. It also ensures the project's condition - moving back to the start block before time ends.
Since the agent only moves in a block after a safe confirmation, it can go into any random block. This fact easily leads to the case the agent is unable to move further and hence, can not earn better score. Considering the case:

(4)	(5)	Breeze(6)	
Start (1)(3)	Breeze (2)		

Now at step 6, the algorithm will guide our agent to shoot below. If we shoot it right:

(4)	(5)	Breeze(6)	
Start (1)(3)	Breeze (2)		

The newly-shooted block is now safe. The agent will continue to move:

(4)	(5)	Breeze(6)	
Start (1)(3)	Breeze (2)	Wumpdied (7)	

Regardless of the next move, the senario will alway be:

(4)	(5)	Breeze(6)	
Start (1)(3)	Breeze (2)	Wumpdied (7)	Breeze
		Breeze	

Although it is totally safe in those green blocks, they are predicted twice to hide a Wumpus. The agent will then, definitely, shoot and lose points.

In conclusion, although the algorithm is simple and work efficiently in most cases, it does not guarantee the optimal score at the end.

ASSIGNMENT PLAN

Start date	End date	Task
29/10	18/11	Implement level 1
18/11	30/11	Learn about Q-learning
1/12	12/12	Implement level 2
12/12	21/12	Fix bug Implement graphical demonstration

ENVIRONMENT TO COMPILE AND RUN OUR PROGRAM

Visual studio 2017, C++

ESTIMATING THE DEGREE OF COMPLETION

No.	Description	Completion
1	Finish level 1	100%
2	Finish level 2	100%
3	Graphical demonstration	100%
4	Generate at least 5 maps	100%
5	Algorithm, experiment and comment report	100%

REFLECTION AND COMMENT

The first difficulty is to generate the game and implement the agent independently so that the agent does not violate any original data from the map.
After finishing the code, we continue to check if the prediction satisfies all collected prerequisites and gives an authentic safe guidance for the agent to move.
Some bugs we get (and luckily fixed) along the way: The agent goes in loop, The agent dies after going to a dead wumpus block, Input file errors.
However, we find this project insightful and informative. Throughout the project, we get to know logical algorithm application in computer science project, the Q-learning algorithm and get a glimpse on how an AI performs to solve problems.

REFERENCES:

- Artificial Intelligence - A Modern Approach, Third Edition
- A Painless Q-Learning, http://mnemstudio.org/path-finding-q-learning-tutorial.htm