**Project 02**

**Wumpus World**

**Team member:**

Nguyễn Vũ Quang Minh 1651065

Phạm Thế Quyền 1651029

1. **Description**

The purpose of this project is to design and implement a logical search agent and AI agent for a partially-observable environment. This will be accomplished by implementing an agent that navigates through the Wumpus World.

1. **New Wumpus World**

We will modify the Wumpus world as such:

* The world will be limited in **10x10**.
* Agent can appear in any Room (**xa**, **ya**) and always facing to the right. This room is the only room have the cave door.
* There may be any number of pits and gold in the world.
* There is at least **one Wumpus**.
* The agent carries **an infinite number of arrows**.
* The agent only has a limited time to explore the rooms, before the cave door collapses and the agent becomes trap inside forever. The agent only has time to visit **150** rooms.

The score are as such:

* **Add 100** points for picking up each **gold**.
* **Reduce** **100** points for shooting an **arrow**.
* **Reduce 10000** points for **dying** (by being eaten by the Wumpus, falling in a pit, or being trapped inside the cave).
* **Add 10** point for **climbing out** of the cave.
* There is **no cost** for **moving** from one room to the next.

**Input:** the given map is represented by matrix, which is stored in the input file, for example, **map1.txt**. The input file format is described as follows:

* The first line contains an integer **N**, which is the size of map.
* N next lines represent the **N × N** map matrix. Each line contains M integers. The number at [i, j] (row i, column j) determines the state of rooms. If room has some things or signal such as **W**umpus, **P**it, **B**reeze, **S**tench, or **A**gent, it is marked by first capitalized character in name of each type and written next to each other. Between two adjacent rooms is separated by a space (“ “).If room empty, it is marked by hyphen character (-).

**Output:**

**Level 1:** : Logical Search Agent

For this level, you must implement code to explore the Wumpus World and get the highest score possible, using First-Order Logic and resolution to solve it.

**Solution:**

We priotirize the moving direction of agent as: Up > Right > Left > Down.

The idea is every time the Agent enters a room, we build knowledge base (KB) based on state of the room the agent is inside.

When checking state of current room, we have two lists:

- The first one is **“Or”** list which stores literals. The “Or” operation will be performed for all literals in that list. For example: **o = [A, B, C]** so we use **o** as **(A ∨ B ∨ C)**.

- The second one is **“And”** list which also stores literals. The “And” operation will be performed for all **negation** of literals in that list. For example: **a = [A, B , C]** so we use **a** as **(~A ∧ ~B ∧ ~C).**

The format of literal’s label is: P\_ij or W\_ij where P presents Pit, W presents Wumpus, (i,j) is the index of current room:

- P\_i,j: This room has pit or not

- W\_i,j: This room has wumpus or not

Any room (i, j) that the agent can currently stand inside is the safe room which does not have Wumpus or Pit inside. That statement is certainly true so we append two literals to **“And”** list: W\_ij and P\_ij immediately before checking a state of the current room.

Otherwise, we gain information of current room and adjacent rooms to update the KB like the following:

**Case 1: room[i][j] = ‘-‘**

This means room (i, j) is safe and we are also sure that Wumpus and Pit are not in adjacent rooms. Therefore, we put all new literals to the “And” list.

We may have a sentence to update KB:

~P\_i+1,j ∧ **~**P\_i-1,j ∧ **~**P\_i,j+1 ∧ **~**P\_i,j-1 ∧

**~**W\_i+1,j ∧ **~**W\_i-1,j ∧ **~**W\_i,j+1 ∧ **~**W\_i,j-1

Notably, we also check the validity of room’s index to put approciate literals to the list. Ant literals whose name has invalid (i, j) wont be put to list. The above sentence is just a full case when 4 rooms next to the current room is valid. Similarly for next cases.

**Case 2: room[i][j] = ‘B’**

This means room (i, j) is also safe but any adjacent rooms may have the Pit and certainly, thoes rooms don’t have Wumpus. Therefore, we put literals related to “P” to Or list and literals related to “W” to And list. We do the same thing with above case for room[i][j] = ‘S’.

Update KB with a sentence:

(P\_i+1,j ∨ P\_i-1,j ∨ P\_i,j+1 ∨ P\_i,j-1) ∧

**~**W\_i+1,j ∧ ~W\_i-1,j ∧ ~W\_i,j+1 ∧ ~W\_i,j-1

With room[i][j] = ‘S’, a sentence used to update KB is:

(W\_i+1,j ∨ W\_i-1,j ∨ W\_i,j+1 ∨ W\_i,j-1) ∧

**~**P\_i+1,j ∧ ~P\_i-1,j ∧ ~P\_i,j+1 ∧ ~P\_i,j-1

**Case 3: room[i][j] = ‘BS’ or room[i][j] == ‘BSG’ (or any swapping of characters)**

This means room(i, j) is safe but any adjacent rooms can contain the Pit or Wumpus and we are not sure about that. Therefore, we put literals of both “W” and “P” to Or list.

Update KB with a sentence:

P\_i+1,j ∨ P\_i-1,j ∨ P\_i,j+1 ∨ P\_i,j-1 ∨

W\_i+1,j ∨ W\_i-1,j ∨ W\_i,j+1 ∨ W\_i,j-1

After that we will have a big list of knowledge base. We have a function to solve this KB and return a list of safe rooms for agent to enter.

We have a BFS function to find the way home from the current room to escape room with the opened room (the rooms have been passed)