VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

CSC14003 – AI PROJECT

WUMPUS WORLD

LECTURER: Mr. Le Ngoc Thanh

Mrs. Ho Thi Thanh Tuyen

GROUP MEMBERS:

Trần Minh Đức	18127027
Nguyễn Vũ Thu Hiền	18127004
Đinh Phi Long	18127263

Ho Chi Minh City, 8/2020

MỤC LỤC

I. INTRODUCTION

1. ABSTRACT

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

It is a game with:

- 2D cave connected by passages.
- Some rooms contain a pit into which we fall and perish.
  - We feel a breeze if near a pit.
- One room contains a Wumpus that will eat us.
  - We have one arrow that we can shoot in the direction we are facing.
  - We smell a stench if near the Wumpus.
- Somewhere, there is a pot of gold.
- We can move forward or backward, turn left by 90 degrees or right by 90 degrees.
- Find gold (if possible) and try and kill the Wumpus.

Wumpus World

- The world will be limited in 10x10. Room (1, 1) will still be the bottom-left one, and Room (10, 10) the top-right one. First number is room position in horizontal coordinate and second number is room position in vertical coordinate.
- Agent can appear in any Room (x_a , y_a) 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 game will end when one of the following three conditions occurs:

  1. The agent dies
  2. The agent kills all of the Wumpus AND grabs all the gold
  3. The agent climbs out of the cave

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).

- Add 10 point for climbing out of the cave.
- Reduce 10 points for moving from one room the the next.

2. ENVIRONMENT

- Language: Python 3.7+
- Graphic Game Framework: Pygame
- Enviroment Software: Anaconda
- Addition library: pytweening (optional)
- OS Build: Window 10 2004 (64 bits)

3. ROLE & ASSIGNMENT

Milestone	Tasks	Date	Assigned to	Progress
Base Game Structure	- Base game structure - Graphic display - Guild, rule to develop	20/08/2020	Đức	100%
	- Map parser, sample search implements base on map	22/08/2020	Đức	100%
Implement problem	- Design assets	22/08/2020	Hiền	100%
	- Problem Analyze	22/08/2020	Long, Hiền	100%
	- KB implement	23/08/2020	Đức	100%
	- Develop Algorithm & Implementations	23/08/2020-24/08/2020	Đức	100%
	- Unit test phase with small problem (4x4)	24/08/2020	Đức	100%
	- BFS to optimize path to a safe position	24/08/2020	Đức	100%
Optimization	- Finished screen	25/08/2020	Hiền	100%
	- Optimize and release	25/08/2020	Đức	100%
Test, Build Map, Report	- Test, Build Map, Re-port	26/08/2020	Hiền, Long	100%
Report	- Algorithm + Result	27/08/2020	Long	100%
	- Others	27/08/2020	Hiền	100%

II. APPROACH SOLVE PROBLEM

- 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 with each line represents a string. If room empty, it is marked by hyphen character (-). 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 dot (.)

	BS	W	BS	P	B				

  ° This row is represented: -.BS.W.BS.P.B.-.-.-.-

- Output: a result with path for agent, game point will be displayed on screen and/or written in output text file such as result1.txt.

ALGORITHM IDEA

In the Knowledge base (KB), we have many lists to store things such as: the cells which are pits; the cells which are wumpus; the pit cells which are solved or not solved; the wumpus cells which are solved or not solved; the visited nodes and the safe nodes.

A cell is safe if and only if that cell does not contain a wumpus or a pit. If a cell has a stench sign then one of the adjacent cells must be wumpus. If a cell has a breeze sign then one of the adjacent cells must be pit. If a cell has both a breeze sign and a stench sign then the adjacent cells be wumpus or pit.

We have many tell() and ask() functions to help the agent check what the state of a cell is.

The agent start at (0, 0) then get all the adjacent cells it can move. Check the states of the cells and make the decision to choose the direction to move throughout perceive() function. If that cell contains a gold, the scores will increase 100 points.

The algorithm will open all the safe nodes (the node that the agent infers is definitely safe). The games will end when all of the safe nodes are open.

III. GAME STRUCTURE

□ src

- assets: The resources of this project, including fonts and images
- cores:

- kb_solve:

    + kb.py: KnowledgeBase

    + agent.py: Agent abstract

    + glu_agent.py: Glucose Agent (base on glucose3 – referenced from HW)

- layout

    + parser.py: Read and process map from file

- search:

    + bfs.py: BFS algorithm to get the shortest path

- agent: necessary object to use in BFS algorithm. This structure base on LAB assignment 1 (search algorithm – 18127027)

- ui: solving graphic of each agent

- wumpus: solving wumpus, manage the title

- maps: including maps to test
- screens: contains EventScreen all screens object inherited from GameScreen.
- states: contains state of game (not playing)
- gpath.py: including global path in project
- setting.py: setting config for the game
- wumpus.py: main game

MAPS: including many maps

IV. GUILD TO USE

- Open workspace at folder "***src***"
- Run "***wumpus.py***"
- If you want to change map in file for tesing, copy a map from ***folder MAPS***, paste it in "***src/maps/map_10_10.txt***". Then, close the current game if it is openning and reopen it.

- The implement is compatitve with any square size of map (4x4, 10x10,…). If you want to test, copy map to file maps/map_10_10.txt

V. RESULT AND MEASURE

- Our application implements all the requirements 1 to 5. We used pygame as graphic framework to show the output and simulate the movement of player

+ The algorithm will visit all safe nodes, that can be use KB to make sure not PIT or WUMPUS

+ We don't visit wumpus and kill them since we don't know the gold of this map. But Wumpus will shown if it 's solved

+ The cost to visit is 10 but gold only 100 so it have many negative scores.

- Created 10 maps with maxium size is 10x10

- **map1 (10x10)**

  Score : -930

  Gold: 4

  Step: 132

  Safe nodes: 79

  Position of all gold is:

  [(3, 3), (4, 1), (8, 8), (2, 2)]

- **map2**

  Score: 1110

  Gold: 27

  Step: 158

Safe nodes: 96

Position of all gold is:

[(3, 5), (3, 6), (4, 6), (4, 5), (4, 1), (2, 1), (2, 3), (2, 7), (2, 8), (2, 9), (1, 9), (1, 8), (1, 1), (1, 0), (5, 5), (5, 1), (6, 1), (7, 5), (7, 4), (7, 1), (8, 1), (8, 5), (9, 4), (9, 3), (9, 2), (9, 1), (9, 0)]

- **map3**

  Score: -990

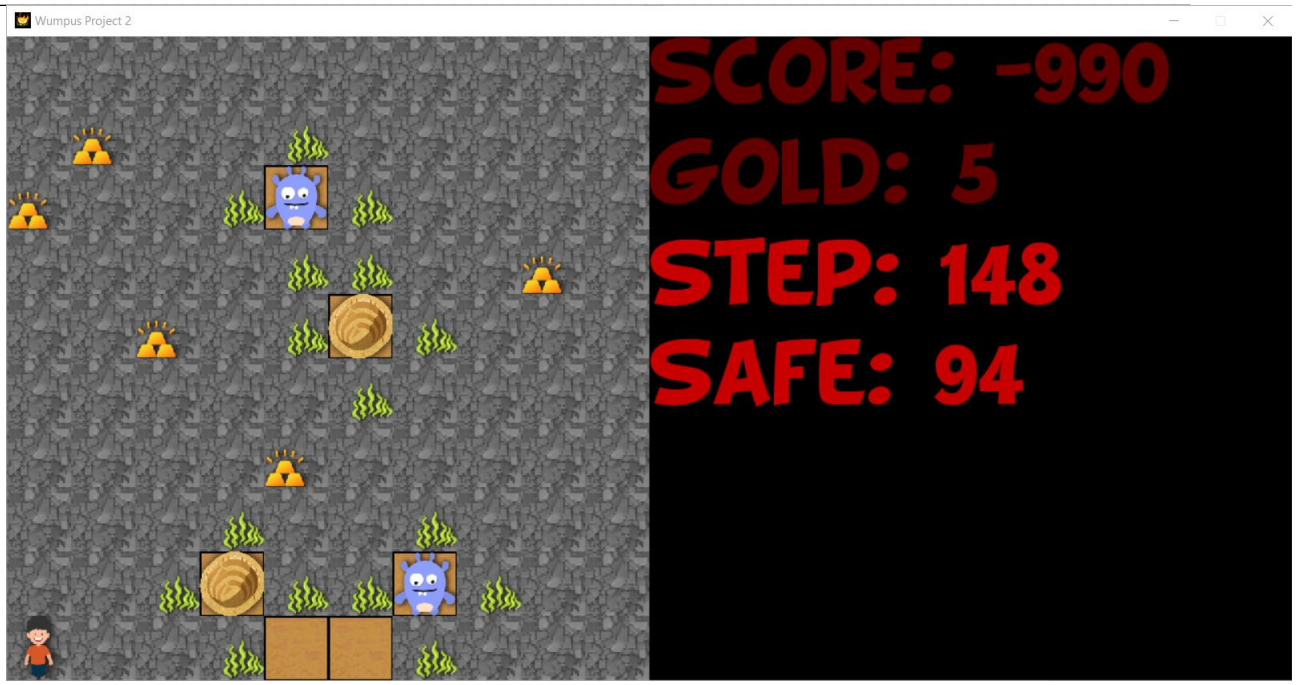
  Gold: 5

  Step: 148

  Safe: 94

  Position of all gold is:

  [(0, 7), (2, 5), (1, 8), (4, 3), (8, 6)]



- **map4**

  Score: -990

  Gold: 6

  Step: 148

  Safe: 88

  Position of all gold is:

  [(0, 1), (1, 3), (3, 4), (5, 6), (3, 7), (4, 5)]

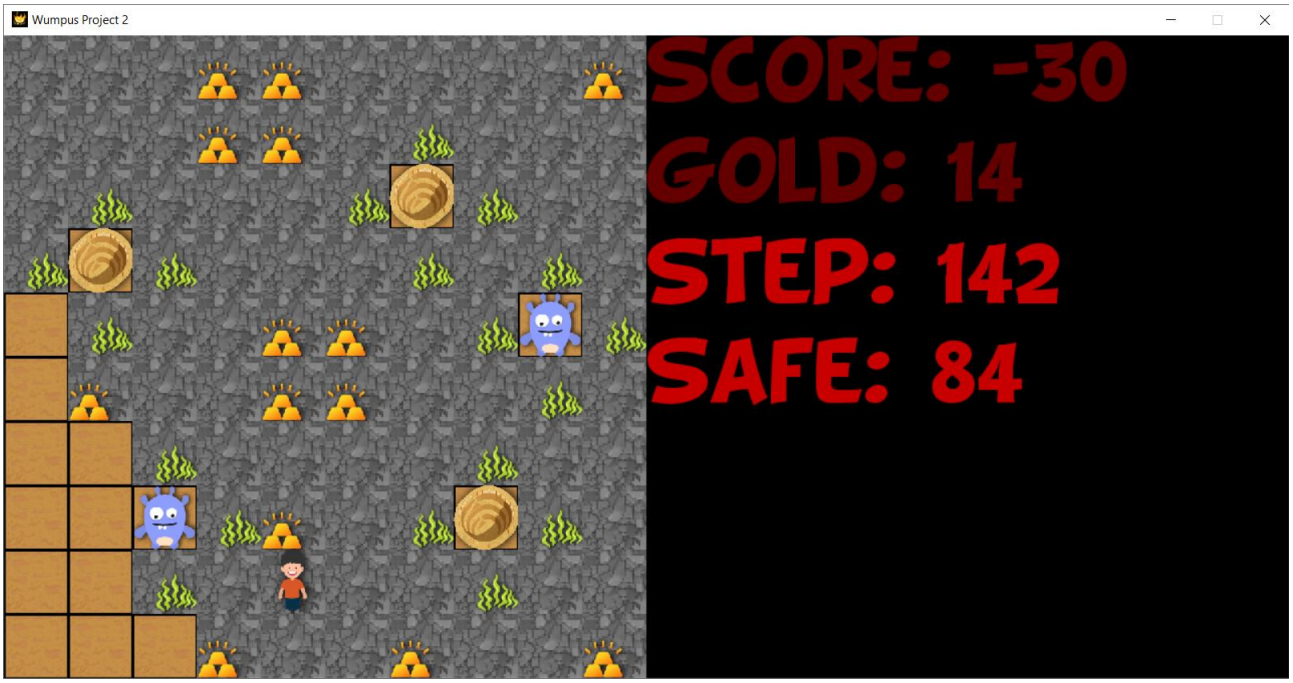
- **map5**

Score: -30

Gold: 14

Step: 142

Safe: 84



- **map6**

  Score: 250

  Gold: 17

  Step: 144

  Safe nodes: 89

- **map7**

  Score: 450

  Gold: 14

  Step: 94

  Safe nodes: 57

- **map8**

  Score: -110

Gold: 14

Step: 160

Safe: 87

Position of all gold is:

[(7, 7), (8, 5), (8, 4), (8, 9), (9, 8), (9, 7), (5, 0), (4, 1), (3, 3), (3, 2), (4, 9), (1, 4), (2, 3), (1, 1), (3, 0)]

- **map9**

  Score: 350

  Gold: 10

  Step: 64

  Safe: 37

  Position of all gold is:

  [(4, 0), (2, 2), (2, 5), (1, 3), (1, 1), (1, 0), (5, 1), (6, 2), (6, 5), (0, 4)]

- **map10**

Score: 470

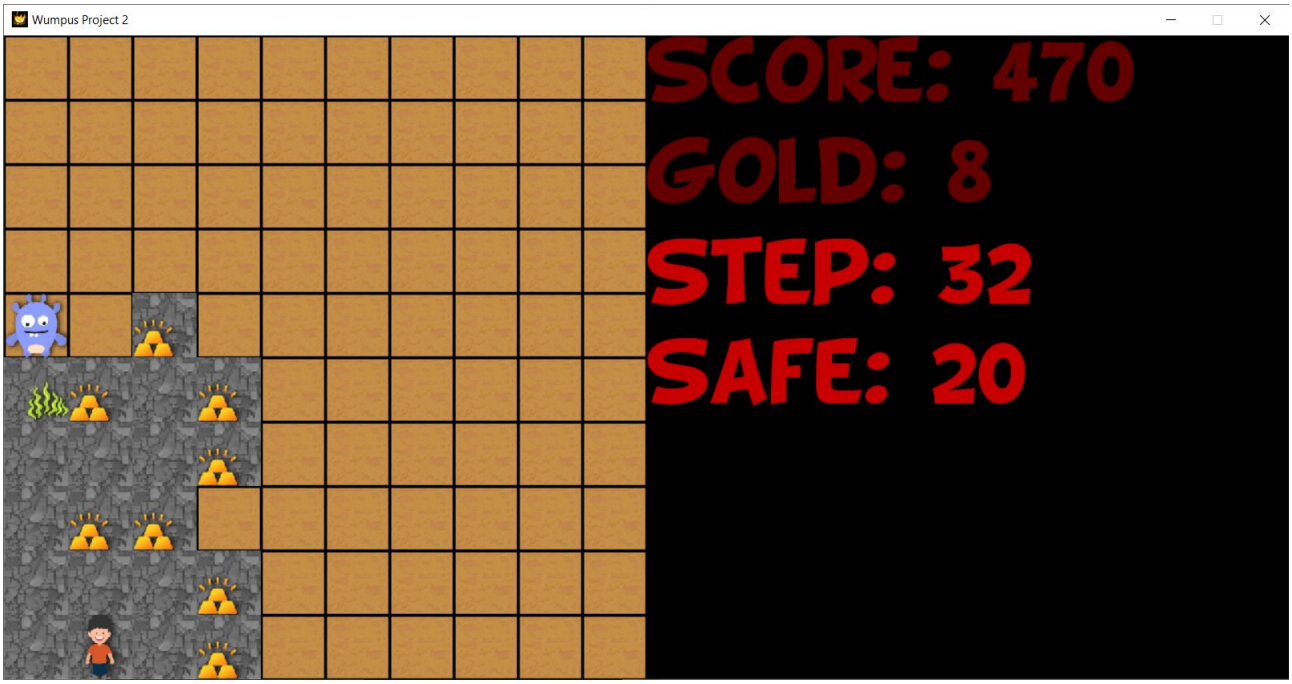
Gold: 8

Step: 32

Safe: 20

Position of all gold is:

[(1, 2), (3, 0), (3, 1), (2, 2), (1, 4), (2, 5), (3, 4), (3, 3)]



VI. REFERENCES

[1]. https://github.com/pysathq/pysat

[2]. https://www.pygame.org/wiki/tutorials